UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

UNISPHERE

RAZTEGLJIVA INFRASTRUKTURA ZA VARNO KOMUNIKACIJO V PORAZDELJENIH SISTEMIH

Jernej Kos

Delo je pripravljeno v skladu s Pravilnikom o podeljevanju Prešernovih nagrad študentom, pod mentorstvom doc. dr. Mojce Ciglarič.

Ljubljana, 2011

Povzetek

V diplomskem delu smo raziskali obstoječe teoretične rešitve na področju usmerjanja z uporabo razteglјivih prekrivnih omrežij, ki temeljijo na principu strukturiranih topologij in komunikacije vsak-z-vsakim. Nato smo na podlagi tega pregleda zasnovali in razvili komunikacijsko ogrodje v programskem jeziku C++, ki omogoča izmenjavo sporočil med aplikacijami na razvijalcu enostaven način. To ogrodje, imenovano UNISPHERE, lajša omrežno programiranje porazdeljenih aplikacij, saj namesto razvijalca skrbi za nizkonivojske podrobnosti, ki so ključne za varno in razteglјivo arhitekturo. To pomeni, da se lahko razvijalec osredotoči na vsebinsko plast, namesto da bi se moral ukvarjati s podrobnostmi komunikacije.

Ključne besede:

porazdeljeni sistemi, komunikacije, varnost, razteglјivost, ogrodje, prekrivno omrežje, arhitektura, C++

Abstract

We have researched existing theoretical solutions in the field of scalable overlay network routing, based on structured topologies and peer-to-peer communications. We have then used this research and knowledge to design and implement a communications framework in the form of a C++ library which enables simple developer-friendly message-based communication between applications. This framework, called UNISPHERE, eases the development of distributed applications, since it handles all the low-level details that are key for a secure and scalable architecture, by itself. This enables the developer to focus herself on application logic instead of wasting time with communications details.

Keywords:

distributed systems, communications, security, scalability, framework, overlay network, architecture, C++

Kazalo

Poglavje 1

Uvod

Današnji svet je vedno bolj povezan, Internet vključuje vedno več naprav in povezuje vedno več ljudi. Vse moderne aplikacijske rešitve morajo izkoristiti to povezanost, kar pomeni, da morajo komunicirati preko prostranih omrežij z drugimi rešitvami, z drugimi sistemi, z drugimi storitvami in z drugimi ljudmi. Sistemi ne smejo imeti kritičnih točk, katerih izpad bi pomenil njihovo sesutje v celoti (angl. single points of failure).

Zato sistemi ne smejo biti centralizirani, biti morajo porazdeljeni. Vsakemu, ki se je že kdaj ukvarjal z omrežnim programiranjem je jasno, da je pisanje porazdeljenih aplikacij nepotrebno oteženo. Obstoječi primitivi, kot so vtičniki, za take aplikacije nikakor niso neposredno ustrezni, ker zahtevajo preveč ročnega dela. Ročno vzdrževanje vseh interakcij porazdeljenega sistema pa zelo hitro lahko postane težavno in drago. Bistvene zahteve iz katerih izvira težavnost gradnje porazdeljenih sistemov so:

- **Razteglјivost *(angl. scalability)*:** Vedno večje število vozlišč, ki jih želimo vključiti v komunikacijski sistem za pisce omrežnih aplikacij predstavlja vedno večjo kompleksnost pri vzdrževanju vseh različnih povezav in protokolov. Prav tako lahko ob nespametnem načrtovanju komunikacijski protokol popolnoma odpove pri topologijah z zelo velikim številom vozlišč.

- **Varnost:** Varnost komunikacij je v današnjem svetu bistvenega pomena. Na žalost poskusi [22] pri popravljanju obstoječega sklada TCP/IP pri zagotavljanju varnosti niso povsem uspešni. Rešitve kot je IPsec so sicer tehnično dobre na področju varnosti, vendar imajo hude pomanjkljivosti, ki preprečujejo preprosto uporabo v novih aplikacijah. Te pomanjkljivosti so povezane predvsem z nezmožnostjo prilagajanja oviram, ki obstajajo v obstoječih omrežnih topologijah (glej naslednjo točko) in z razvijalcem

neprijaznimi implementacijami zaradi kompleksnosti samega sistema. Po drugi strani je obstoječa infrastruktura javnih ključev popolnoma neprimerno zasnovana, saj uporabniki vse zaupanje implicitno polagajo v roke podjetij in državnih institucij, ki nudijo storitve certifikatnih agencij. To zaupanje pa velikokrat ni povsem upravičeno. Alternativa se ponuja v obliki DNSSEC [3], vendar tudi ta sistem ne predstavlja bistvenega napredka, ker je zaupanje v osnovi še vedno centralizirano. V zadnjem času se tako jasno kaže, da potrebujemo nov, bolj decentraliziran in torej porazdeljen pristop k zaupanju.

- **Univerzalna povezljivost:** Internetne topologije so zelo raznovrstne. Požarni zidovi, prevajalniki omrežnih naslovov (angl. network address translators, NATs) in druge ovire preprečujejo univerzalno povezljivost med vsakim parom vozlišč IP. Postopna uvedba protokola IPv6 morda res ponovno omogoča možnost takšne povezljivosti, vendar bodo vedno obstajali razlogi zaradi katerih simetrična povezljivost ne bo vedno mogoča. To dejstvo še dodatno oteži razvoj porazdeljenih sistemov.

- **Mobilnost:** Nekatera vozlišča porazdeljenega sistema lahko niso statična. To pomeni, da se kontaktni podatki vozlišča (naslov IP) lahko spremenijo. Zanašanje na predeterminirane naslove za celotno porazdeljeno infrastrukturo tako ni primerno, ker to zahteva dodatno vzdrževanje. Infrastruktura mora omogočati samodejno odkrivanje lokacij storitev v omrežju.

- **Enostavnost uporabe:** Osnovni gradniki TCP/IP, bolj natančno vtičniki POSIX [1], so preveč nizkonivojsko orodje za uporabo v modernih komunikacijskih sistemih. To pomeni, da mora razvijalec veliko funkcionalnosti razviti sam. Po drugi strani je veliko komunikacijskih ogrodij pretirano kompleksnih za razvijalca oz. ne ponuja prave porazdeljene funkcionalnosti.

Gradnja porazdeljenih sistemov pa ni problematična zgolj s stališča komunikacije, vsaka aplikacija ima še vrsto svojevrstnih problemov. Komunikacijsko ogrodje seveda ne more rešiti teh aplikacijsko specifičnih težav, lahko pa omogoči razvijalcu, da se osredotoči nanje in da mu ni potrebno skrbeti še za težave s samo komunikacijo. S perspektive uporabnika ogrodja mora torej le-to rešiti dve stvari: omogočati mora komunikacijo med poljubnimi aplikacijami na poljubnih vozliščih v omrežju in to storiti na tak način, ki je za razvijalca preprosto razumljiv in enostaven za uporabo.

1.1 Opredelitev problema

Če torej povzamemo in strnemo uvodne besede, lahko problem definiramo z naslednjima točkama:

1. Za razvoj večjih porazdeljenih sistemov imamo trenutno na voljo različne paradigme (arhitekture), različne tehnološke rešitve in različna orodja, ki pa vsa zahtevajo veliko znanja o sami strukturi porazdeljenih sistemov in njihovih usmerjevalnih protokolih.

2. Posledično je učna krivulja za razvijalce zelo visoka, saj se morajo seznaniti z vso teorijo, ki stoji za učinkovitimi rešitvami, in njeno optimalno implementacijo. V nasprotnem primeru tvegajo slabše rešitve tako s teoretičnega kot s tehničnega vidika.

To je problem, katerega reševanja smo se lotili v našem delu.

1.2 Področje raziskave

Naše raziskovanje je v tem diplomskem delu izviralo iz abstraktnega modela porazdeljenega sistema, na podlagi katerega smo postavili vprašanje ali je mogoče načrtovati in realizirati takšno arhitekturo, ki bo z enotnim ogrodjem podprla razvoj večjih porazdeljenih sistemov in se tako čim bolj približala rešitvi omenjenega problema.

Prvi cilj je torej zasnova in implementacija ogrodja, ki bo s svojimi zahtevami primerno za uporabo znotraj zaprte organizacije, ki za komunikacijo na nivoju IP uporablja obstoječa lokalna in prostrana omrežja. Drugi, prav tako pomemben, cilj pa je teoretsko osvetliti določene dele področja porazdeljenih sistemov in porazdeljenega zaupanja, ki se zdijo pri razvoju še posebej pomembni in v katera bi bilo dobro usmeriti nadaljnje raziskave.

Kot dokaz in potrditev naše hipoteze smo na podlagi naših raziskav implementirali takšno ogrodje in z njegovo pomočjo razvili preprost prototip porazdeljene aplikacije za potrebe testiranja. Rezultati diplomskega dela so večplastni, na eni strani je to modularna in razširljiva implementacija komunikacijskega ogrodja za porazdeljene sisteme na principu komunikacije vsak-z-vsakim (angl. peer-to-peer), na drugi strani pa teoretična osvetlitev nekaterih še ne povsem rešenih problemov na področju porazdeljenga zaupanja in varnosti v porazdeljenih sistemih.

1.3 Uporabna vrednost predvidenih rezultatov

Rezultati v obliki komunikacijskega ogrodja odpirajo možnost razvoja vrsti porazdeljenih aplikacij. Sem spada celotno področje omrežnega upravljanja, kar zajema npr. porazdeljene požarne zidove, ki si lahko izmenjujejo informacije med vozlišči in ki jih je moč konfigurirati za celotno omrežje iz ene lokacije. Prav tako sem spada konfiguracija strežnikov oz. naprav na enoten način in iz ene lokacije, naprave pa lahko same odkrijejo kakšna konfiguracija je namenjena njim in jo udejanjijo.

Sistem je mogoče uporabiti tudi za razvoj porazdeljene avtentikacijske infrastrukture, ki omogoča avtenticiranje uporabnikov na omrežje in uporabo storitev na njem. Hkrati je možno razvite module nadgraditi s podporo avtorizaciji in porazdeljenem zaupanju, kar prinaša spet nove možnosti pri sodelovanju med aplikacijami. Takšna porazdeljena infrastruktura je tudi nadvse primerna za upravljanje storitev v oblačnih arhitekturah, kjer se strežniške instance lahko dinamično dodajajo ali odvzemajo, potrebuje pa se komunikacijska infrastruktura za povezavo med vsemi temi instancami.

1.4 Organizacija diplomske naloge

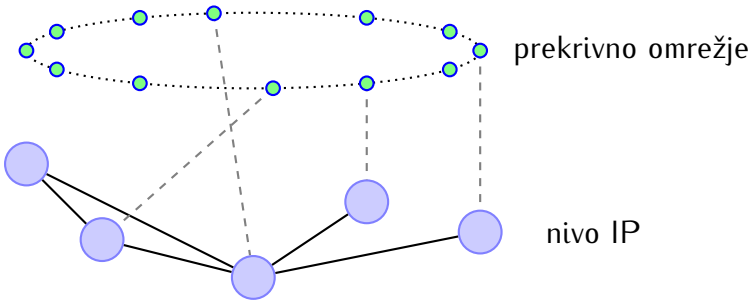
Drugo poglavje predstavlja teoretični pregled enega izmed ključnih konceptov, ki omogoča izgradnjo porazdeljenih sistemov - prekrivnih omrežij. Sledi poglavje o varnosti, ki izpostavlja nekatere varnostne problematike v takšnih sistemih ter poda delne teoretične rešitve zanje. Četrto poglavje se nato osredotoči na opis konkretne arhitekture s konkretnimi rešitvami in protokoli. Razloži prilagoditve in razširitve enega izmed obstoječih prekrivnih omrežij za potrebe učinkovitega usmerjanja v restriktivnih omrežnih topologijah. V detajle opiše delovanje posameznih komponent modularnega komunikacijskega ogrodja, ki je implementirano v obliki zbirke C++ knjižnic. Sledi poglavje z rezultati testiranja te konkretne implementacije, nato pa kratek zaključek s povzetkom možnih aplikacij ogrodja ter načrti za nadaljnje delo.

Poglavje 2

Pregled protokolov prekrivnih omrežij

Usmerjanje prometa poteka v omrežjih IP na podlagi identifikatorjev imenovanih naslovi IP. Ti naslovi so 32-bitna (IPv4) oz. 128-bitna (IPv6) števila. Ker želimo zasnovati porazdeljeni sistem, ki bo neodvisen od tega, kje se vozlišča nahajajo v topologiji IP, je potrebno uvesti novo vrsto identifikatorjev. To takoj pomeni, da potrebujemo tudi novo vrsto usmerjevalnih protokolov. Termin, ki zajema takšna omrežja, ki gradijo nad protokoli TCP/IP (oz. teoretično nad poljubnim drugim omrežnim nivojem), je *prekrivno omrežje* (angl. overlay network). V nadaljevanju bo izraz prekrivno omrežje pomenil prekrivno omrežje nad protokoli TCP/IP.

Prekrivna omrežja (slika 2.1) omogočajo gradnjo decentraliziranih porazdeljenih sistemov, ki se znajo samoorganizirati v primerne topologije. Njihovi koncepti omogočajo tudi gradnjo usmerjevalnih protokolov, ki jih bomo potrebovali za arhitekturo našega komunikacijskega ogrodja. Zato je pomembno, da se najprej seznanimo s teorijo, ki stoji za njimi.



Slika 2.1: Koncept prekrivnega omrežja nad nivojem IP.

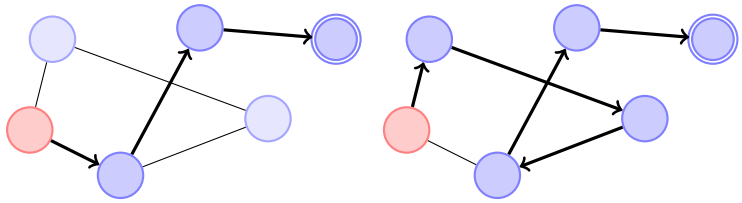
2.1 Strukturirana in nestrukturirana

Konceptualno obstajata dve vrsti prekrivnih omrežij, strukturirana in nestrukturirana. Kot nakazuje že ime, nestrukturirana prekrivna omrežja ne predpostavijo nobene v naprej določene topologije za samo omrežje. To pomeni, da se vozlišča pri usmerjanju sporočil ne morejo zanašati na znano topologijo in da je potrebno uporabljati tehnike kot so naključni sprehodi (angl. random walks), poplavljanje grafa (angl. graph flooding) ali pa katerega izmed usmerjevalnih protokolov iz sveta IP. Težava poplavljanja je seveda v tem, da lahko generira veliko količino odvečnega prometa po omrežju in tako naredi komunikacijo neučinkovito. Težava usmerjevalnih protokolov (tako *link-state* kot *distance-vector* in *path-vector*), ki so znani iz usmerjanja na nivoju IP (npr. OSPF [26]) pa je v tem, da zahtevajo $O(n)$ prostora za usmerjevalne tabele pri n vozliščih. To pa predstavlja skrajno nezaželjeno lastnost pri velikem številu vozlišč.

Na drugi strani strukturirana omrežja organizirajo vozlišča v neko vnaprej določeno topologijo na podlagi unikatnih identifikatorjev vozlišč. Ker je topologija prekrivnega omrežja znana, jo lahko vozlišča izkoristijo za veliko bolj optimalno usmerjanje sporočil (kot bomo videli v nadaljevanju, le-ta navadno zagotavljajo usmerjanje sporočil v največ $O(\log n)$ korakih pri vzdrževanju $O(\log n)$ stanja v usmerjevalni tabeli na vozlišče). To seveda hkrati pomeni, da je topologija omejena in da so sosedi vsakega vozlišča lahko samo točno določena vozlišča, odvisno od njihovih identifikatorjev.

Za primerjavo obeh konceptov prekrivnih omrežij s stališča usmerjanja je potrebno definirati metriko imenovano *razteg* (angl. stretch). Razteg pomeni razmerje med številom korakov (vozlišč) na poti, ki jo generira nek usmerjevalni protokol in številom korakov v najkrajši možni poti med vozliščema. Razteg bomo opazovali na topologiji IP, ki se nahaja pod prekrivnim omrežjem. Razteg, ki je enak 1 pomeni, da algoritem najde najkrajše možne poti, vrednosti večje od 1 pa pomenijo manj optimalne poti (za boljšo ilustracijo glej sliko 2.2).

Na področju izboljšanja usmerjanja v nestrukturiranih omrežjih je kar nekaj raziskav, npr. Ganesan in sodelavci [17] v svojem članku predstavijo protokol, ki ga imenujejo YAPPERS in združuje nekatere prednosti nestrukturiranih in strukturiranih omrežij. Podobno Chawather in sodelavci [8] predstavijo izboljšave protokola Gnutella, tako da pri usmerjanju poizvedb in grajenju povezav upoštevajo kapaciteto posameznih vozlišč. Bolj zanimive so raziskave na področju kompaktnega usmerjanja, Abraham in sodelavci [2] predstavijo usmerjevalno shemo, ki zagotavlja navzgor omejen razteg (največ 3) pri prostorski zahtevnosti $O(\sqrt{n \log n})$ na vozlišče. Do podobnega rezultata pridejo tudi Singla in sodelavci [33], ki s pomočjo protokolov kompaktnega usmerjanja dosegajo relativno



(a) $r = 1.0$ (b) $r = 1.67$

Slika 2.2: Pomen raztega pri usmerjanju sporočil skozi omrežje na določeni topologiji. Rdeče vozlišče je izvor sporočila, puščice kažejo pot sporočila, dvojno obkroženo vozlišče je cilj.

nizke raztege pri usmerjanju sporočil.

2.1.1 Kratka zgodovina

Sodobna prekrivna omrežja imajo svoj začetek v aplikacijah za deljenje datotek, ki delujejo na principu povezave enak z enakim (angl. peer-to-peer). Nastala so, da bi odpravila centraliziranost storitev za izmenjavo glasbe, saj je se je to v primeru aplikacije Napster izkazalo kot ključna pomanjkljivost - zaradi pravnih sporov so bili prisiljeni izklopiti strežnike, ki so uporabnikom posredovali naslove IP drugih uporabnikov in omogočali iskanje po datotekah, ki so jih uporabniki dajali na voljo. Zaradi centralizirane narave indeksiranja in distribucije naslovov IP je bila s tem celotna storitev onemogočena.

Kot rešitev teh težav se je pojavila Gnutella, ki implementira nestrukturirano prekrivno omrežje in je popolnoma decentralizirana. Vozlišča se organizirajo v naključno topologijo in uporabljajo poplavljanje ter naključne sprehode za iskanje drugih vozlišč in podatkov. Težava s takim pristopom je, kot že omenjeno, nerazteglјivost celotnega sistema. Pri vedno večjem številu vozlišč naključno usmerjanje iskanj generira velike količine nepotrebnega mrežnega prometa, kar omeji uporabno velikost omrežja.

Raziskave na področju porazdeljenih razpršenih tabel (angl. distributed hash tables, DHTs) so dale okoli leta 2001 in po njem vrsto strukturiranih prekrivnih omrežij, med njimi so Pastry [30], Tapestry [40], Chord [36], CAN [29], Skipnet [19] in Kademlia [24]. Zaradi strukturiranega pristopa se takšna omrežja veliko bolje obnesejo za iskanje in distribucijo podatkov o lokaciji deljenih datotek. Vendar, ali so strukturirana prekrivna omrežja primerna tudi za usmerjanje sporočil za potrebe ogrodja za porazdeljeno komunikacijo? Usmerjanje je v svetu IP domena nestrukturiranih topologij, razteglјivost pa se zanaša na hierarhično

delitev naslovov IP in njihovo združevanje (angl. prefix aggregation). Ta hierarhična delitev pa se v modernih omrežjih vedno bolj podira, ker vedno bolj potrebujemo prenosljiv (angl. provider-independent, PI) naslovni prostor za potrebe redundance in mobilnosti med ponudniki. To pa povzroča razdrobljenost prostora in eksponentno rast usmerjevalnih tabel. Zaradi tega pri načrtovanju novih arhitektur ne moremo spregledati eksponentne rasti in moramo načrtovati z mislijo na porazdeljene algoritme, ki prinašajo podlinearno razteglјivost.

Zanimiv koncept takšne arhitekture je predstavljen v delu avtorja B. Forda, UIA (angl. Unmanaged Internet Architecture) [14], kjer so analizirani različni pristopi pri porazdeljenem usmerjanju. Članek predstavi arhitekturo za komunikacijo med osebnimi napravami uporabnikov, ki mora delovati brez prisotnosti centralnih entitet in tudi samo z lokalno povezljivostjo med napravami. Za usmerjanje avtor preuči tako porazdeljene razpršene tabele, kompaktno usmerjanje kot tudi usmerjanje z omejenim poplavljanjem po lokalnem družabnem omrežju.

Vsekakor obstajajo razlogi za in razlogi proti uporabi strukturiranih omrežij za usmerjanje, v nadaljevanju bomo preučili oboje.

2.1.2 Prednosti in pomanjkljivosti

Očitna prednost strukturiranih omrežij je $O(\log n)$ poraba prostora za usmerjevalno tabelo na vsakem vozlišču. To pomeni, da tudi ob visoki rasti števila vozlišč v prekrivnem omrežju ostaja število potrebnih vnosov v tabelah precej majhno. Prav tako strukturirana omrežja zaradi svoje organizacije večinoma zagotavljajo $O(\log n)$ v številu korakov, ki so potrebni za dostavo sporočil iz vozlišča A v vozlišče B. Na drugi strani pri nestrukturiranih omrežjih, ki uporabljajo tehniko poplavljanja grafa, nimamo takih zagotovil, kar pomeni da lahko v določenih primerih pride do velikih količin nepotrebnega prometa zaradi samega poplavljanja. Boljšo možnost predstavlja kompaktno usmerjanje, najnovejši protokoli na tem področju zahtevajo $O(\sqrt{n \log n})$ vnosov v usmerjevalni tabeli [33]. Slaba stran kompaktnega usmerjanja so v praksi še relativno nepreizkušeni in zapleteni protokoli.

Vendar nič ni zastonj in težava strukturiranih prekrivnih omrežij se pokaže, ko pogledamo razteg na nivoju IP. Ker je topologija le-teh v naprej omejena, morajo vozlišča vzpostavljati povezave z vnaprej določenimi sosedi glede na razdaljo med njihovimi identifikatorji (oz. podobnim merilom, odvisno od protokola). Identifikatorji so v strukturiranih omrežjih po prostoru razporejeni naključno (to zagotavlja generiranje identifikatorjev s pomočjo kriptografske zgoščevalne funkcije, npr. SHA-1), saj je ravno to tista lastnost, ki omogoča uravnoteženost

in logaritemsko razteglјivost. To seveda pomeni, da bližina v prostoru identifikatorjev prekrivnega omrežja nima nobene korelacije z bližino (in posledično z raztegom) na nivoju IP. Zaradi tega lahko na videz kratke poti v prekrivnem omrežju (razteg 1, če opazujemo topologijo prekrivnega omrežja) v resnici pomenijo dolge poti preko celotnega omrežja IP (visok razteg, če opazujemo topologijo omrežja IP), čeprav sta morda vozlišči, ki poskušata komunicirati, fizično blizu. To težavo se da omiliti (vendar brez matematično dokazanih zagotovil) s pomočjo določene fleksibilnosti v usmerjevalnih tabelah in izbiro sosedov na podlagi zunanje metrike kot je npr. število korakov na nivoju IP oz. merjenja RTT[1]. Ta metoda bo tudi podrobneje opisana v nadaljevanju.

Kljub temu, da imajo nestrukturirana omrežja nekatere prednosti in da najnovejše raziskave na področju kompaktnega usmerjanja obljubljajo relativno razteglјive rešitve, se bomo v nadaljevanju osredotočili zgolj na strukturirana omrežja. Razlog za to je, da se za usmerjanje še vedno deloma zanašamo na nivo IP, kjer imajo usmerjevalni algoritmi nizek razteg (običajno 1). Vseeno bo pojem raztega prišel prav, ko bo potrebno premostiti nekatere težave, ki jih moderne internetne topologije povzročajo strukturiranim omrežjem, opisane v razdelku 4.1.1 na strani 45.

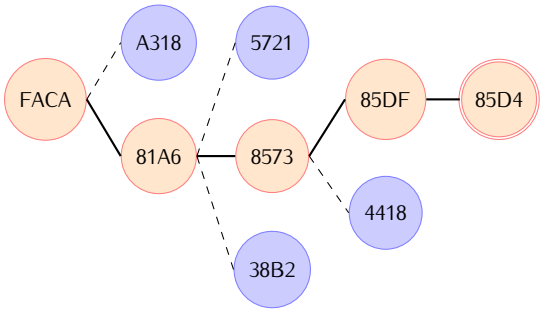
Na tem mestu je potrebno omeniti še zanimiv članek [20], v katerem Jain in sodelavci predlagajo uporabo strukturiranega prostora identifikatorjev za usmerjanje neposredno na nivoju IP. V članku opišejo način kako s pomočjo strukturiranega pristopa razbremenijo usmerjevalnike, hkrati pa omogočijo tudi ločitev naslavljanja od samega usmerjanja. Seveda je potrebno tukaj zelo dobro paziti na razteg, saj se tukaj ne moremo več zanašati na optimalnost nižjih protokolov usmerjanja in tudi nimamo matematičnih zagotovil o nizkem raztegu (za razliko od prej omenjenih nestrukturiranih protokolov kompaktnega usmerjanja).

2.2 Pregled strukturiranih protokolov

Načini strukturiranja topologije prekrivnega omrežja se od protokola do protokola močno razlikujejo. Kar pa je skupno vsem so operacije, ki se lahko izvajajo nad takim omrežjem. To so:

- **Najdi vozlišče:** Ta operacija je namenjena razreševanju kontaktnih podatkov vozlišč iz notranjih identifikatorjev, ki jih uporablja prekrivno omrežje, v zunanje identifikatorje (naslove IP), ki jih uporablja nivo IP. To razreševanje je ključno za uspešno usmerjanje podatkov po prekrivnem omrežju.

[1] *Round Trip Time*, to je čas, ki preteče med pošiljanjem zahtevka in prejetjem odgovora



Slika 2.3: Usmerjanje v Plaxtonovi mreži.

- **Najdi vrednost:** Strukturirani protokoli so zasnovani kot porazdeljene razpršene tabele, ki omogočajo shranjevanje parov *(ključ, vrednost)* in kasnejšo poizvedbo po tej vrednosti. Ključi za dostop do vrednosti se generirajo ponavadi na enak način kot identifikatorji vozlišč, torej s pomočjo zgoščevalne funkcije kot je npr. SHA-1.

- **Shrani vrednost:** Shrani par *(ključ, vrednost)* v porazdeljeno razpršeno tabelo.

Nekaj primerov takšnih protokolov sledi v nadaljevanju. Pri vsakem protokolu je opisan način strukturiranja topologije in prednosti oz. slabosti, ki jih ima ta način v primerjavi z ostalimi protokoli. Pri tem se bomo osredotočili predvsem na usmerjanje sporočil in manj na shranjevanje in lokacijo objektov v porazdeljeno razpršeno tabelo.

2.2.1 Tapestry

Protokol Tapestry [40] temelji na geometriji Plaxtonove mreže (angl. Plaxton mesh) [27]. Vsako vozlišče ima dodeljeno unikatno število - identifikator vozlišča - v številski bazi b. Usmerjanje poteka na podlagi teh identifikatorjev in sicer tako, da se na vsakem koraku ciljnemu identifikatorju približamo za eno števko. Za primer recimo, da obsega prostor identifikatorjev števila dolžine $d = 4$ v bazi $b = 16$ (kar da velikost prostora identifikatorjev vozlišč 16^4). Izvorno vozlišče je `FACA`, ciljno vozlišče pa je `85D4` (glej sliko 2.3). Pot skozi graf prekrivnega omrežja gre v tem primeru po vrsti (razrešujemo števke od leve proti desni) `FACA` $\Rightarrow$ `8***` $\Rightarrow$ `85**` $\Rightarrow$ `85D*` $\Rightarrow$ `85D4`. Zvezdica v tem primeru pomeni katerokoli števko (torej na prvem koraku bi lahko vozlišče `FACA` usmerilo sporočilo kateremukoli vozlišču, ki se začne s števko `8`, na drugem koraku kateremukoli vozlišču,

ki se začne s številkama 85 in tako naprej), kar pomeni, da naredijo prvi koraki velikanske skoke čez prostor identifikatorjev, kasnejši pa vedno manjše.

Preslikava sosedov

Vsako vozlišče Plaxtonove mreže vzdržuje preslikavo sosedov (angl. neighbour map). To preslikavo si lahko zamislimo kot raztreseno matriko (angl. sparse matrix) kazalcev s toliko stolpci kot je dolžina identifkatorjev, torej po en stolpec za vsako številko v identifikatorju vozlišča. i-ti stolpec vsebuje kazalce na kontaktne podatke za tista vozlišča, ki si delijo z lokalnim vozliščem vsaj i prvih številk. Torej 3. stolpec v preslikavi sosedov za vozlišče `FACA` vsebuje kazalce za vozlišča `FA`$**$, vrstice v tem stolpcu predstavljajo identifikatorje `FA0`$*$, `FA1`$*$, ..., `FAF`$*$. Število vrstic je torej odvisno od izbrane baze b. Matrika za vozlišče `FACA` bi izgledala takole:

$$\begin{bmatrix} c_{0***} & c_{F0**} & c_{FA0*} & c_{FAC0} \\ c_{1***} & c_{F1**} & c_{FA1*} & c_{FAC1} \\ c_{2***} & c_{F2**} & c_{FA2*} & c_{FAC2} \\ \vdots & \vdots & \vdots & \vdots \\ c_{F***} & c_{FF**} & c_{FAF*} & c_{FACF} \end{bmatrix}$$

Vrednost c_{id*} v i-tem stolpcu predstavlja kontaktne podatke za poljubno vozlišče, kjer se prvih i številk sosednjega vozlišča ujema s prvimi $i-1$ številkami lokalnega vozlišča in z zadnjo številko v predponi id. Ker so identifikatorji vozlišč generirani z neko zgoščevalno funkcijo in tako zelo razpršeni po prostoru, je velik del take matrike prazen v omrežjih, kjer je $n \ll b^d$.

Kot nakazuje zgornja konstrukcija, nam takšna organizacija (ob predpostavki konsistentnih preslikav sosedov) zagotavlja največ $\log_b n$ korakov pri usmerjanju skozi prekrivno omrežje velikosti n vozlišč ob izbrani bazi b. Usmerjanje skozi Plaxtonovo mrežo si lahko predstavljamo kot sprehod po vpetem drevesu s korenom na izvornem vozlišču. Na i-tem koraku si vozlišče s ciljnim vozliščem deli vsaj i številk, v naslednjem koraku pa vozlišče preveri vnos v matriki, kjer se ujema $i+1$ številk in s tem zmanjša prostor iskanja za b-krat.

Nadomestno usmerjanje

Preden se lotimo opisa protokolov za vzdrževanje prekrivnega omrežja, je potrebno opisati še postopek nadomestnega usmerjanja (angl. surrogate routing). Gre za usmerjanje v primeru neobstoječih ciljnih identifikatorjev. V tem primeru se mora sporočilo konsistentno usmeriti proti nekemu vozlišču, ne glede na to

iz katerega dela omrežja izvira. Za ta namen predpostavlja Plaxtonova shema globalno relacijo urejenosti vozlišč in statično omrežje. Tapestry usmerja sporočila kot sicer, dokler ne naleti na prazne vnose v preslikavi sosedov (če ciljni identifikator ne obstaja, mora po prej omenjeni konstrukciji obstajati vsaj en tak prazen vnos na poti do cilja).

Ko postopek usmerjanja naleti na tak prazen vnos, se mora odločiti za neko drugo povezavo in sicer na globalno konsistenten način. To pomeni, da bo usmerjanje po katerikoli poti vedno pripeljalo do enakega nadomestnega vozlišča. Za to se v Tapestry uporablja deterministični algoritem za izbiro vozlišč, ki izbira med tistimi, ki so v danem stolpcu preslikave sosedov neprazni. Usmerjanje se ustavi, ko edini neprazen vnos pripada vozlišču samemu (to se lahko seveda zgodi zgolj v zadnjem stolpcu). To vozlišče je potem nadomestno vozlišče za dani neobstoječi ciljni identifikator.

Tak način sicer lahko povzroči podaljšanje poti, vendar v [40, stran 9] avtorji pojasnjujejo, da je takšno podaljšanje zanemarljivo.

Vzdrževanje omrežja ob njegovi dinamičnosti

Plaxtonova shema sama po sebi predpostavlja predhodno globalno znanje o vseh vozliščih v omrežju. To s perspektive dinamičnega okolja, kjer se vozlišča lahko prekrivnemu omrežju pridružujejo in odhajajo, ni pretirano uporabno. Bistvo Tapestry je torej razširitev te sheme s protokoli in algoritmi za dinamično vzdrževanje prekrivnega omrežja.

Za primer vzemimo, da imamo vozlišče z identifikatorjem A, ki se želi vstaviti v omrežje. Pri prihodu novega vozlišča se preko nekega obstoječega vozlišča usmeri sporočilo s ciljnim identifikatorjem A. Pot gre skozi več vozlišč, vozlišče, ki ga obiščemo na i-item koraku označimo z N_i. Na vsakem koraku vozlišče A prenese i-ti stolpec preslikovalne matrike z vozlišča N_i in ga uporabi za i-ti stolpec svoje preslikovalne matrike. Ker ima vsaka celica v preslikovalni matriki več možnih kontaktov, na tem mestu vozlišče A naredi lokalno optimizacijo za izbor tistega kontakta, ki se bo uporabljal za primarno usmerjanje. To stori na podlagi zunanjih metrik, kot je npr. že omenjeni RTT. Ta optimizacija je uporabna zato, da se usmeri sporočila skozi najbližje povezave in se s tem doseže nižji razteg na nivoju IP. Postopek optimizacije lahko uporablja tudi sosede odkritih sosedov, da ima večjo izbiro. Ta optimizacija se lahko vrši v ozadju, saj ni nujno potrebna za delovanje, omogoča pa hitrejši pretok podatkov pri usmerjanju. Polnitev preslikovalne matrike se konča, ko vozlišče A opazi prazen vnos za naslednji korak.

Sedaj ko ima vozlišče A zgrajeno svojo preslikovalno matriko, mora le-to ob-

vestiti druga relevantna vozlišča o svojem prihodu, tako da lahko tudi druga vozlišča posodobijo svoje preslikovalne matrike na ustreznih mestih. Ta postopek uporablja vzvratne kazalce nadomestnega usmerjanja za obvestila relevantnih vozlišč. Zaradi teh posodobitev je lahko postopek vstavljanja novega vozlišča relativno drag, zato Tapestry pri nepričakovanem odhodu vozlišč ne izvede takoj popolnega izbrisa vozlišč ampak jih zgolj označi kot nedosegljiva, odstrani pa jih šele po preteku daljšega časovnega obdobja.

Zaradi dragega vstavljanja vozlišč, je Tapestry neprimeren za omrežja, kjer se vozlišča zelo veliko in hitro premikajo, to npr. drži za senzorska omrežja. Za področje senzorskih omrežij si lahko bralec ogleda rešitev usmerjanja, predstavljeno v [39].

2.2.2 Pastry

Podobno kot Tapestry tudi Pastry [30] za usmerjanje sporočil uporablja geometrijo Plaxtonove mreže. Usmerjanje torej poteka na približno enak način, na vsakem koraku se ciljnemu vozlišču približamo za 1 števko v bazi b. Poleg organizacije v Plaxtonovo mrežo se v Pastry uporablja tudi neke vrste dvojno povezan seznam v listih že prej omenjenega vpetega drevesa - za ta seznam se uporablja izraz množica listov (angl. leaf set).

Vsako vozlišče v množici listov vsebuje nekaj kazalcev na predhodna in nekaj kazalcev na sledeča vozlišča. Algoritem za usmerjanje na vsakem koraku najprej preveri ali ciljni identifikator pade znotraj intervala, ki ga pokriva množica listov in ga v tem primeru dostavi vozlišču katerega identifikator mu je najbližji po številski vrednosti. V primeru, da interval množice listov ne vsebuje ciljnega identifikatorja, se uporabi standardni postopek usmerjanja kot je opisan že pri Tapestry.

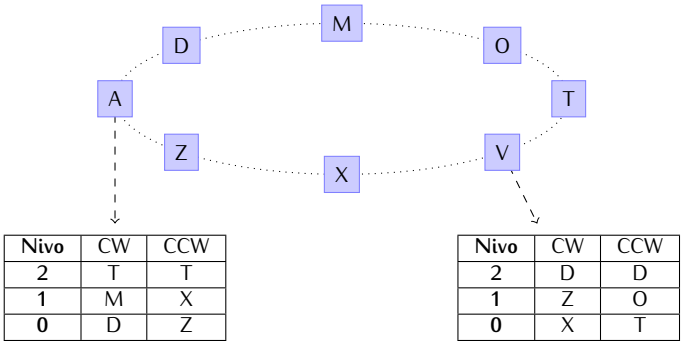
Usmerjanje proti vozlišču z najbližjo številsko vrednostjo se v Pastry uporablja kot rešitev za nadomestno usmerjanje, torej v primeru da ciljno vozlišče ne obstaja. Uporaba dveh različnih metrik za usmerjanje v resnici predstavlja težavo, ki zakomplicira usmerjanje in ima za rezultat manj optimalne poti, ker je razdalja po vsaki izmed metrik drugačna [24, stran 2]. To težavo in dvofazno usmerjanje (navadno ter nadomestno) ima tudi Tapestry.

Protokoli za vzdrževanje omrežja

Protokol za vstavljanje novega vozlišča v prekrivno omrežje se od Tapestry ne razlikuje bistveno. Razlika je v tem, da mora novo vozlišče pridobiti tudi ustrezne množice listov, ki jih lahko pridobi iz kontaktov v nižjih nivojih na novo zgrajene

preslikovalne matrike. Za popravilo vnosov izpadlih vozlišč v svoji množici listov pa Pastry zahteva množice listov od ostalih vozlišč, ki so še dosegljiva. Ta postopek zagotavlja, da bo usmerjanje delovalo, razen v kolikor hkrati odpove več kot polovica vozlišč v množici listov. Zaradi tega je pri implementaciji potrebno ustrezno določiti velikost te množice.

2.2.3 SkipNet



Nivo	CW	CCW
2	T	T
1	M	X
0	D	Z

Nivo	CW	CCW
2	D	D
1	Z	O
0	X	T

Slika 2.4: Vozlišča v prekrivnem omrežju SkipNet, urejena po imenu. Prikazani usmerjevalni tabeli za vozlišči A in V.

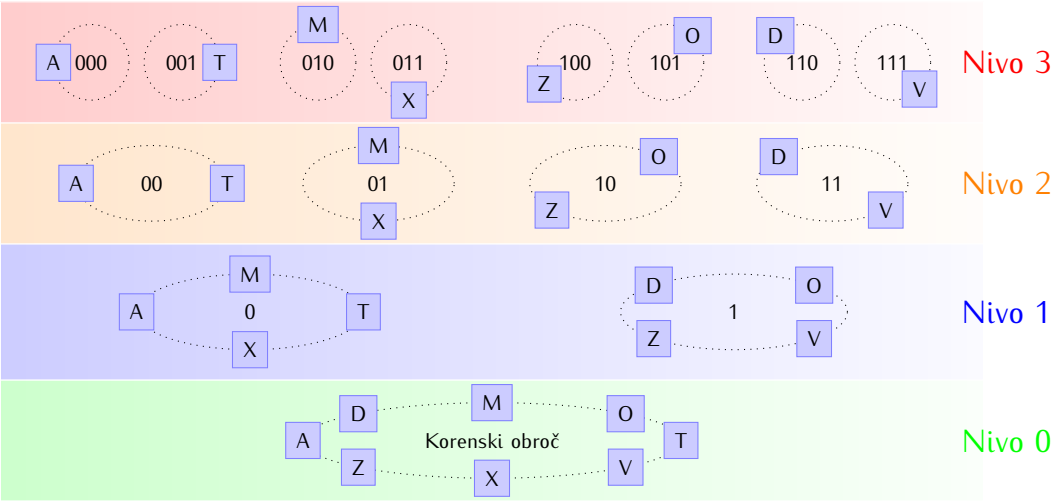
Zasnova prekrivnega omrežja SkipNet [19] se močno razlikuje od prej omenjenih. Osnovno vodilo pri zasnovi tega omrežja je bilo doseči možnost administrativne določitve lokalnosti vozlišč in objektov shranjenih na njih. Pri vseh drugih konceptih DHT so identifikatorji naključno razpršeni po omrežju, tako da je kakršnakoli lokalnost na podlagi identifikatorjev izgubljena. Prav tako zaradi razpršenosti identifikatorjev usmerjanje med vozlišči znotraj organizacije večinoma pomeni, da bodo poti vodile skozi vozlišča zunaj organizacije, kar lahko ni zaželjeno.

Koncept, ki ga uporablja SkipNet, izhaja iz podatkovne strukture preskakujočega seznama (angl. skip list) [28]. Ta struktura se sicer uporablja za učinkovito iskanje vrednosti (časovna zahtevnost reda $O(\log n)$, podobno kot uravnotežena dvojiška drevesa).

Struktura in identifikatorji

Omrežje SkipNet uporablja pri usmerjanju dve vrsti identifkatorjev, imenske in številske. Imenski identifikatorji so navadni ASCII nizi poljubne dolžine.

Usmerjevalna tabela omrežja SkipNet je podobno kot preskakujoči seznam sestavljena iz več nivojev (glej sliko 2.4). Na i-tem nivoju tabela vsebuje kazalce na vozlišča, ki so približno 2^i korakov oddaljeni od lokalnega vozlišča in sicer za obe smeri[2] (kot dvojno povezan seznam na vsakem izmed nivojev). Če za boljšo predstavo združimo vse usmerjevalne tabele vseh vozlišč na vseh nivojih, dobimo več obročev, kot so vidni na sliki 2.5. Vsako vozlišče na tej sliki ima v svoji usmerjevalni tabeli na vsakem nivoju dve sosednji vozlišči (po eno za vsako smer).



Slika 2.5: Obroči vseh nivojev usmerjevalne strukture SkipNet.

V primeru, da bi se striktno držali strukture, da kazalec na i-tem nivoju preskoči ravno $2^i - 1$ vozlišč, bi bila naša struktura enakovredna *popolnemu* preskakujočemu seznamu. Takšen popoln seznam pa ima pomanjkljivost in sicer vstavljanja vanj so zelo draga [28, stran 1]. Podobno, kot obstaja rešitev za preskakujoče sezname na podlagi verjetnostne določitve povezav znotraj seznama, tudi SkipNet uporablja verjetnost. Vsak izmed obročev na i-tem nivoju se razdeli v dva obroča na na nivoju $i+1$ (kot kaže slika 2.5). Vendar to razdeljevanje ni vedno enako, temveč se vsako vozlišče naključno (z enakomerno porazdelitvijo) odloči kateremu izmed dveh možnih obročev se bo pridružilo. Na tak način vstavitve novih vozlišč vplivajo zgolj na dve vozlišči v vsakem obroču v katerem je vozlišče, ki se vstavlja. Kot pri preskakujočih seznamih, tudi tukaj takšna verjetnostna porazdelitev vozlišč še vedno omogoča usmerjanje v $O(\log n)$ korakih z visoko verjetnostjo [19, stran 5].

[2]Na sliki se pojavljata dve oznaki smeri: CW, *clockwise* - v smeri urinega kazalca; CCW, *counter-clockwise* - nasproti smeri urinega kazalca

Izbira obroča na vsakem nivoju se lahko predstavi kot dvojiška vrednost (`0` oz. `1`). Kot je to vidno iz slike 2.5 (nivo 3), dobimo na ta način številske identifkatorje vozlišč. To je drug tip identifikatorjev, ki jih uporablja SkipNet. Prvih i bitov številskega identifikatorja predstavlja izbiro obroča na i-tem nivoju, generirati pa jih je mogoče, kot omenjeno že pri ostalih prekrivnih omrežjih, s pomočjo uporabe zgoščevalne funkcije npr. SHA-1 (pomembno je zgolj, da so identifikatorji unikatni in naključno porazdeljeni) nad imenskim identifikatorjem vozlišča.

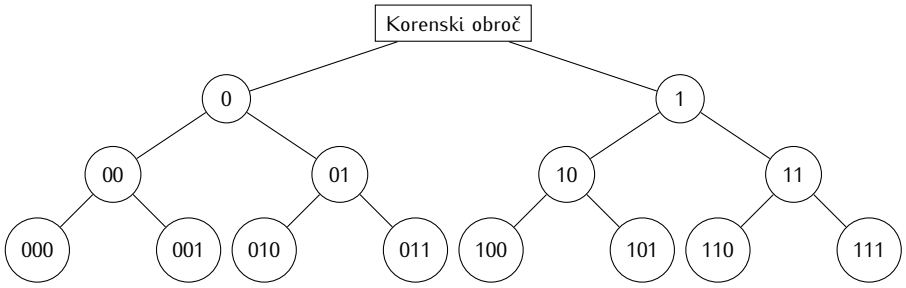
Usmerjanje po imenskih identifikatorjih

Ker SkipNet uporablja dve vrsti identifikatorjev, potrebuje dva usmerjevalna algoritma. Algoritem za usmerjanje po imenskih identifikatorjih deluje podobno kot iskanje v preskakujočih seznamih. Na vsakem izmed vozlišč po nivojih usmerjevalne tabele preverimo na kateri interval spada ciljno vozlišče in sicer tako, da preskočimo kar največ (začnemo torej na najvišjem nivoju) v ustrezni smeri. Če za primer vzamemo, da se nahajamo v vozlišču `A` in želimo poslati sporočilo vozlišču `V`, v lokalni usmerjevalni tabeli najprej pogledamo na nivo 2. V primeru, da bi imeli vozlišči v imenskem identifikatorju skupno predpono, bi bila smer usmerjanja pomembna (usmerjanje mora iti v taki smeri, da ostane znotraj predpone, ki ponavadi pomeni oznako organizacije). V našem primeru skupne predpone ni, tako da lahko upoštevamo katerokoli smer.

Edini vnos (glej sliko 2.5) na tem nivoju je vnos za vozlišče `T` in le-to leži po relaciji urejenosti nizov med `A` in `V`. Zato sporočilo posredujemo vozlišču `T`. To vozlišče prav tako začne na drugem nivoju, edini vnos, ki se tam nahaja, je vnos za vozlišče `A`. Ker `A` ne leži med `T` in `V`, se premaknemo en nivo nižje. Tam imamo kazalce na vozlišči `M` in `X`, nobeno izmed njih prav tako ne leži med lokalnim in ciljnim vozliščem. Zato gremo še en nivo nižje, na najnižji nivo, kjer imamo kazalca na `O` in `V`. Našli smo ciljno vozlišče `V` in mu zato posredujemo sporočilo. Kot že omenjeno je število korakov pri usmerjanju po imenskih identifikatorjih $O(\log n)$ (z visoko verjetnostjo, za dokaz glej [19, stran 20]).

Usmerjanje po številskih identifikatorjih

Poleg usmerjanja po imenskih identifikatorjih, je v omrežju SkipNet mogoče tudi učinkovito usmerjanje po številskih identifikatorjih. Ta način usmerjanja je podoben usmerjanju v Tapestry in Pastry v številski bazi $b = 2$, kjer se približujemo cilju po številkah. V strukturi SkipNet ima vsaka številka še pripisan pomen in sicer številka določa obroč na ustreznem nivoju. Zato približevanje po številkah v resnici pomeni premikanje navzgor po nivojih.



Slika 2.6: Obroči strukture SkipNet tvorijo drevo predpon.

Algoritem na vsakem izmed obročov poskuša najti vozlišče, ki se ujema v čim več začetnih števkah in se tako dvigniti na višji nivo. Za primer vzemimo, da sporočilo izvira v vozlišču `A` (`000`), želimo pa ga poslati vozlišču s številskim identifikatorjem `101`. Vozlišče `A` je na prvem nivoju v obroču `0`, ciljno vozlišče pa v obroču `1`, zato se sporočilo posreduje sosedu v korenskem obroču, vozlišču `D` (enako bi veljalo za vozlišče `Z`). Vozlišče `D` ima s ciljnim vozliščem skupno zgolj predpono `1`, v drugi števki pa se razlikuje. Zato posreduje sporočilo sosednjemu vozlišču v prvem nivoju, vozlišču `0`. To vozlišče prejme sporočilo in vidi, da se lokalni številski identifikator ujema s ciljnim identifikatorjem v sporočilu, cilj je torej dosežen. V primeru, da ni vozlišča s točno tem ciljnim identifikatorjem, se sporočilo usmeri k vozlišču, ki je po številski vrednosti najbližje.

Število korakov je tudi pri usmerjanju po številskih identifikatorjih $O(\log n)$ (ponovno, z visoko verjetnostjo, ker je resnična časovna zahtevnost odvisna od porazdelitve identifikatorjev).

Iz opisov obeh algoritmov lahko opazimo, da je učinkovito usmerjanje po imenskih identifikatorjih možno zaradi organizacije imen v porazdeljen preskakujoč seznam (nivoji v strukturi SkipNet v tem primeru predstavljajo nivoje preskakujočega seznama). Hkrati je struktura SkipNet organizirana tudi v drevo predpon (angl. trie) za številske identifikatorje. Če preoblikujemo sliko 2.5 v drevesno strukturo, dobimo drevo predpon na sliki 2.6.

Takšno porazdeljeno drevo predpon pa omogoča učinkovito usmerjanje po številskih identifikatorjih vozlišč.

Vstavljanje in odhod vozlišč

Vstavljanje vozlišča se začne z iskanjem ustreznega najvišjenivojskega obroča, ki ga določa številski identifikator na novo pridruženega vozlišča. To iskanje poteka preprosto z uporabo algoritma za usmerjanje po številskih identifikatorjih. Nato novo vozlišče na najvišjem nivoju poišče svoje sosede s pomočjo

usmerjanja proti svojemu imenskemu identifikatorju, ter s pomočjo odkritih sosedov ta postopek iskanja sosedov ponovi rekurzivno za vse nivoje do korenskega. Ko algoritem pride do korenskega nivoja, obvesti vse na novo odkrite sosede (na ustreznih nivojih), da naj popravijo svoje kazalce in tako vstavijo vozlišče v prekrivno omrežje.

Za zagotavljanje redundance v primeru odhoda vozlišč SkipNet, podobno kot Pastry, uporablja množico listov in sicer na korenskem obroču. Vsako vozlišče hrani torej za vsako smer ne samo enega, ampak več (odvisno od določene dolžine množice listov) sosedov. V primeru izpadov lahko le-te uporabi za popravilo usmerjevalnih tabel. Pomembna lastnost strukture SkipNet je, da je za pravilno delovanje usmerjanja potreben zgolj konsistenten korenski obroč. Višjenivojski obroči se zato lahko popravijo v ozadju - to seveda pomeni, da bo razteg na nivoju prekrivnega omrežja med popravljanjem višji kot ob optimalnih usmerjevalnih tabelah.

Prednosti in težave

Prednost pred ostalimi implementacijami porazdeljenih razpršenih tabel je možnost nadzora lokalnosti. Vozlišča, ki si po imenskih identifikatorjih delijo določene predpone, bodo v strukturi SkipNet skupaj. Prav tako bo imensko usmerjanje med vozlišči z enako predpono vedno obiskalo samo vozlišča znotraj te predpone. V kolikor imena določajo organizacije, to efektivno pomeni, da bodo sporočila potovala zgolj znotraj organizacije. To je koristno tudi pri izpadih, saj izpadi vozlišč zunaj dane predpone/organizacije ne vplivajo na usmerjanje znotraj le-te. Prav tako je to zelo uporabna lastnost s stališča varnosti, saj omejuje napade Sybil (glej razdelek 3.3 na strani 34).

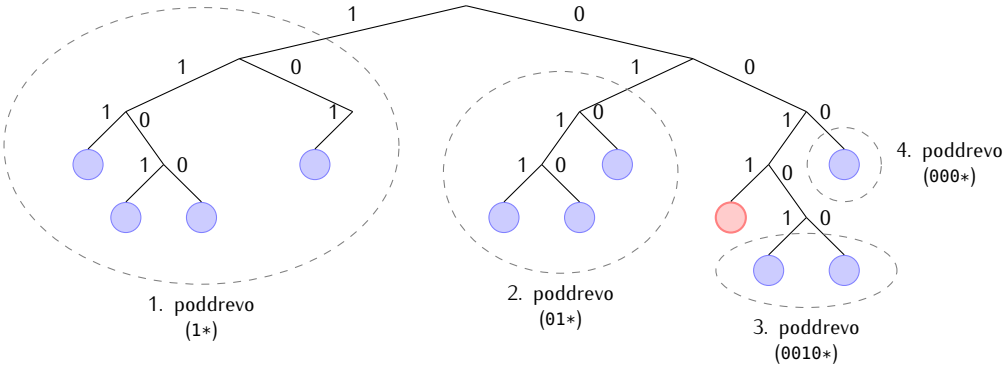
Uporaba imenskih identifikatorjev prav tako pomeni, da so poleg poizvedb po natančni vrednosti ključa objekta možne tudi intervalne poizvedbe (angl. range queries). Ta lastnost lahko pride prav pri iskanju virov.

Žal ima SkipNet tudi nekaj slabih lastnosti, predvsem težaven algoritem za spajanje omrežij v kolikor pride do razdelitve zaradi izpadov [19, stran 16].

2.2.4 Kademlia

Za konec se vrnimo k malce enostavnejšemu prekrivnemu omrežju, ki ga bomo v nekoliko spremenjeni obliki uporabili tudi v našem komunikacijskem ogrodju.

Kademlia [24] je prekrivno omrežje s posebno lepimi matematičnimi lastnostmi, ki izvirajo iz uporabe metrike XOR (ekskluzivni logični ali) za razdaljo med posameznimi vozlišči prekrivnega omrežja. Če primerjamo strukturo omrežja



Slika 2.7: Organizacija prekrivnega omrežja Kademlia v dvojiško drevo, lokalno vozlišče `00110` označeno z rdečo.

s prej obdelanimi protokoli, lahko opazimo veliko podobnosti (pri primerjavi s SkipNet moramo primerjati zgolj številske identifikatorje, saj so imenski njegova posebnost). Identifikatorji vozlišč so pri omrežju Kademlia prav tako rezultat neke zgoščevalne funkcije, ki zagotavlja razpršitev po prostoru identifikatorjev. Uporaba metrike XOR za razdaljo med identifikatorji omogoča enoten algoritem za usmerjanje k poljubnemu identifikatorju - če to primerjamo s Pastry in Tapestry, oba uporabljata ločeno fazo za usmerjanje proti neobstoječim vozliščem, kar nepotrebno kompliciра protokol, usmerjevalne tabele in tudi otežuje formalno matematično analizo omrežja. Vozlišča so v omrežju Kademlia organizirana v dvojiško drevo, kot je prikazano na sliki 2.7.

To drevo spominja na drevo predpon iz prekrivnega omrežja SkipNet (glej sliko 2.6), tudi v tem primeru gre za predpone. Za uspešno usmerjanje mora lokalno vozlišče poznati vsaj eno vozlišče v vsakem izmed poddreves, ki lokalnega vozlišča ne vsebujejo (na sliki 2.7 so ta poddrevesa obkrožena). Usmerjanje v taki strukturi poteka s približevanjem po predponah, podobno kot pri ostalih omrežjih.

Za primer vzemimo, da vozlišče `00110` želi poslati sporočilo vozlišču `11000`. Ker se identifikator ciljnega vozlišča začne s številko `1`, glede na shemo iz slike 2.7 to pomeni, da le-to spada nekam v prvo poddrevo (predpona `1∗`). Predpostavimo, da lokalno vozlišče pozna kontaktne podatke vsaj enege vozlišča iz tega poddrevesa (to zagotavlja protokol Kademlia, kot bomo videli v nadaljevanju) npr. `10100`. Vozlišče `10100` uporablja enako konstrukcijo, vendar seveda drugačno razporeditev poddreves (za to vozlišče prvo poddrevo vsebuje najbolj leva 3 vozlišča na sliki). Podobno kot prej, ker ciljni identifikator vsebuje predpono `11`, algoritem ve, da mora sporočilo usmeriti nekemu vozlišču znotraj tega

poddrevesa, recimo vozlišču `11010`. Eno izmed poddreves tega vozlišča (bolj natančno, drugo poddrevo z leve) pa vsebuje zgolj ciljno vozlišče, torej se sporočilo posreduje `11000` in prispe na cilj.

Metrika XOR

Vsi identifikatorji vozlišč v omrežju Kademlia sestavljajo prostor identifikatorjev. Pri uporabi 160-bitnih identifikatorjev, ki so rezultat zgoščevalne funkcije SHA-1, ima ta prostor obseg od `000...000` do `111...111` (160 števk v dvojiškem zapisu). Omenjena konstrukcija dvojiškega drevesa s slike 2.7 nad tem prostorom določa metriko XOR kot merilo razdalje med dvema vozliščema. Vsa števila moramo jemati po pravilu debelega konca (angl. big endian), tako da najbolj levi bit predstavlja najbolj pomemben bit.

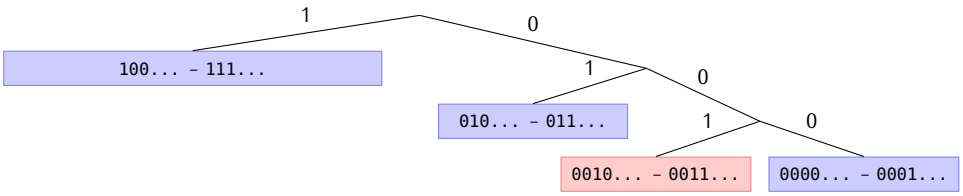
Če uporabimo metriko XOR nad našim drevesom s primera, lahko opazimo, da naravno zajame koncept oddaljenosti od lokalnega vozlišča. Čim manj bitov z leve je skupnih v predponi identifikatorjev, tem bolj narazen sta identifikatorja po metriki XOR in tem bolj narazen sta vozlišči v drevesu predpon. Identifikatorja z dolgo podobno predpono (enakost v veliko prvih bitih) pa da krajšo razdaljo. Takšna mera razdalje nam pomaga pri konsistentni izbiri nadomestnega vozlišča v primeru usmerjanja proti neobstoječim identifikatorjem (nadomestno vozlišče je tisto z najmanjšo razdaljo do ciljnega identifikatorja). To omogoča enoten usmerjevalni algoritem, ki zajame oba primera (primarno in nadomestno usmerjanje).

k-vedra in usmerjevalna tabela

Na sliki 2.7 smo označili poddrevesa v katerih mora lokalno vozlišče poznati kontaktne podatke vsaj enega vozlišča. Vsako tako poddrevo se v terminologiji Kademlie imenuje k-vedro (angl. k-bucket). Gre za seznam dolžine največ k elementov, ki vsebuje kontaktne podatke vozlišč, ki pokrivajo določen razpon identifikatorjev v celotnem prostoru (recimo k-vedro za prvo poddrevo s slike pokriva razpon od `100...000` do `111...111` - polovico celotnega prostora). Da lahko lokalno vozlišče konsistentno usmerja sporočila proti kateremukoli drugemu vozlišču v prekrivnem omrežju, mora imeti k-vedra, ki pokrivajo celoten prostor identifikatorjev.

Posamezna k-vedra so v usmerjevalni tabeli Kademlie organizirana v drevesno strukturo, ki raste organsko, s spoznavanjem novih kontaktnih podatkov drugih vozlišč. Na začetku je celotno drevo zgrajeno samo iz korena, ki vsebuje eno samo k-vedro. To k-vedro pokriva celoten prostor identifikatorjev in

vsebuje lokalno vozlišče. Ko lokalno vozlišče ob spozna še kakšno drugo vozlišče, ga doda v ustrezno k-vedro (v tisto, ki pokriva prostor identifikatorjev, ki vsebuje vozlišče, ki se dodaja). V kolikor pa je ciljno k-vedro polno in vsebuje lokalno vozlišče, ga je potrebno razdeliti. V primeru, da je vedro polno, ampak ne vsebuje lokalnega identifikatorja, je potrebno nek obstoječi element iz njega izbrisati (ali pa zavreči novo vozlišče). Za določitev kaj zavreči, se vozlišča sortira glede na to, kdaj so bila nazadnje odzivna. Primer strukture k-veder za drevo s slike 2.7 si lahko ogledamo na sliki 2.8.



Slika 2.8: K-vedra usmerjevalne tabele v prekrivnem omrežju Kademlia. K-vedro, ki vsebuje lokalno vozlišče je označeno z rdečo. Vsa k-vedra skupaj pokrivajo celoten prostor identifikatorjev brez prekrivanja.

Na tej sliki je razdeljevanje k-veder prišlo do tretjega nivoja. V kolikor bi se napolnilo k-vedro označeno z rdečo, bi se le-to lahko še naprej razdeljevalo. Ta konstrukcija kaže, da za polovico omrežja hranimo zgolj k kontaktnih podatkov drugih vozlišč, prav tako za četrtino omrežja in tako naprej. Nižja vedra bodo seveda vedno bolj prazna, saj bo vedno težje najti vozlišča, ki bi se prilegala vanje (zaradi psevdonaključnosti zgoščevalne funkcije pri dodeljevanju identifikatorjev, so le-ti zelo dobro razpršeni po prostoru in verjetnost, da se prvih i števk v identifikatorju ujema z lokalnim vozliščem, strmo pada z večanjem i).

Manjša težava nastane pri zelo neuravnovešenih drevesih (za uravnovešenost se zanašamo na razprševanje uporabljene zgoščevalne funkcije). Vsako vozlišče mora poznati svojo k-okolico (najbližjih k vozlišč po metriki XOR) zaradi pravilnega usmerjanja k neobstoječim identifikatorjem (k-okolica je analogna množici listov v Pastry). Zaradi strukture drevesa se lahko zgodi, da bi dodajanje novega vozlišča v bratsko vozlišče pomenilo izgubo kontakta iz k-okolice. V tem primeru je potrebno vseeno obdržati k-okolico in razdeliti ciljno k-vedro, kljub temu, da lokalnega vozlišča ne vsebuje. Ker je k-okolica konstantne dolžine k, se v tem primeru usmerjevalna tabela poveča zgolj za konstantni faktor.

Večja vrednost parametra k nam omogoča večjo redundanco v primeru izpadov vozlišč (avtorji protokola priporočajo vrednost $k = 20$).

Optimizacija usmerjanja

Uporabna optimizacija usmerjanja, ki smo jo uporabili tudi v naši implementaciji, omogoča zmanjšanje števila korakov pri usmerjanju v topologiji prekrivnega omrežja. To je mogoče doseči s preprosto spremembo in sicer, da razdeljujemo tudi k-vedra, ki ne vsebujejo lokalnega vozlišča do neke globine b. To omogoča usmerjanje, kjer preskočimo večji del prostora naenkrat (saj imamo bolj natančne usmerjevalne tabele). Seveda večanje b pomeni eksponentno povečanje usmerjevalnih tabel z eksponentom b, zato je potrebno ta parameter določiti pametno (avtorji priporočajo vrednost $b = 5$). Bolj natančno se velikost usmerjevalne tabele poveča na $O(2^b \log_{2^b} n)$ k-veder, medtem ko se pričakovana dolžina poti v prekrivnem omrežju zmanjša na $O(\log_{2^b} n)$ korakov (brez optimizacije $O(\log_b n)$).

Vstavljanje in odhod vozlišč

Vstavljanje vozlišč je v dani strukturi zelo preprosto. Kot že omenjeno se usmerjevalna tabela samodejno polni, ko vozlišča izvejo za druga vozlišča. Vstavljanje se prične s povezavo na neko obstoječe vozlišče B. Vozlišče B doda identifikator ter kontaktne podatke za novo pridruženo vozlišče v ustrezno k-vedro. Nato na novo pridruženo vozlišče izvede poizvedbo za svoj identifikator. Ta poizvedba povzroči, da vsa vozlišča na poti dodajo na novo pridruženo vozlišče v svoje usmerjevalne tabele v ustrezna k-vedra, hkrati pa na novo pridruženo vozlišče pridobi vnose za zapolnitev svojih usmerjevalnih tabel. Nato mora na novo pridruženo vozlišče zgolj osvežiti svoja ravnokar zgrajena k-vedra, ki ne vsebujejo lokalnega vozlišča, in sicer tako, da za vsako k-vedro izvede poizvedbo naključnega identifikatorja, ki pade v dano k-vedro. Osveževanje ponovno služi dvema funkcijama, in sicer omogoča, da na novo pridruženo vozlišče pridobi kontaktne podatke za zapolnitev svojih k-veder, prav tako pa tudi obvesti druga vozlišča o kontaktnih podatkih novega vozlišča.

Prednosti

Prednost prekrivnega omrežja Kademlia izhaja iz njegove konstrukcije in uporabe metrike XOR. Le-ta nam omogoča poenostavljen usmerjevalni algoritem (v primerjavi z ostalimi prekrivnimi omrežji), hkrati pa določeno mero fleksibilnosti pri izbiri sosedov. Največja fleksibilnost je seveda možna pri prvih skokih skozi omrežje, torej pri izbiri kontaktnih podatkov vozlišč, ki spadajo v k-vedra na višjih nivojih. Preprosta in jasna konstrukcija ter optimalno usmerjanje (na topologiji prekrivnega omrežja) je dobra osnova za uporabo v našem komunikacijskem ogrodju. Za uporabo pri usmerjanju sporočil bo potrebno še nekaj

dodatkov, več o le-teh pa v nadaljevanju.

2.3 Oddajanje skupinam na aplikacijskem nivoju

Kot omenjeno že v uvodu, mora fleksibilno komunikacijsko ogrodje omogočati samodejno odkrivanje storitev, katerega implementacija zahteva učinkovito uporabo tehnike oddajanja skupini prejemnikov (angl. multicast). Oddajanje skupini prejemnikov je način komunikacije, kjer se zainteresirana vozlišča lahko naročijo na sporočila pod identifikatorjem neke skupine, v katero nato pošiljatelji sporočila pošiljajo. Za takšne sisteme se uporablja izraz objavi/naroči (angl. publish/subscribe oz. krajše pub/sub). To se razlikuje od bolj običajne komunikacije z enim vozliščem (angl. unicast), kjer poteka komunikacija zgolj med dvema udeležencema.

Tudi neposredno na omrežjih IP že dolgo časa obstajajo protokoli za usmerjanje sporočil skupinam prejemnikov (npr. PIM-SM [12]), ki se veliko uporabljajo za usmerjanje multimedijskih tokov v sistemih IPTV. V tem primeru se za identifikatorje skupin uporabljajo posebni naslovi IP, ki so del rezerviranih območij (na IPv4 je to `224.0.0.0/4`, na IPv6 pa `ff00::/8`).

Težava uporabe oddajanja skupinam na nivoju IP je, da uporaba le-te zahteva postavitev ustrezne infrastrukture na vseh usmerjevalnikih prometa IP (nivo 3 po modelu OSI), kjer bo potekala izmenjava sporočil. Infrastruktura v tem primeru pomeni poganjanje ustrezne programske opreme za usmerjanje, ki implementira ustrezen protokol (kot je npr. zgoraj omenjeni PIM-SM). Na žalost trenutno ni splošno sprejetega modela postavitve oddajanja skupinam v globalnem internetnem omrežju, tako da je edina rešitev implementacija oddajanja skupinam na aplikacijskem nivoju (angl. application-level multicast).

Implementacije na aplikacijskem nivoju se lahko lotimo na veliko načinov, vendar, ker smo se v našem primeru osredotočili na strukturirana prekrivna omrežja, je najbolj smiselna rešitev takšna, ki za oddajanje sporočil skupinam prejemnikov čim bolj učinkovito uporablja kar obstoječe prekrivno omrežje.

2.3.1 SCRIBE

Možna rešitev se ponuja v obliki protokola SCRIBE [6]. V originalnem članku je protokol postavljen nad primitivi, ki jih implementira prekrivno omrežje Pastry, mi pa smo ga v nadaljevanju prilagodili uporabi nad prekrivnim omrežjem Kademlia. Potrebne prilagoditve so minimalne, saj obe prekrivni omrežji ponujata zelo podobne lastnosti.

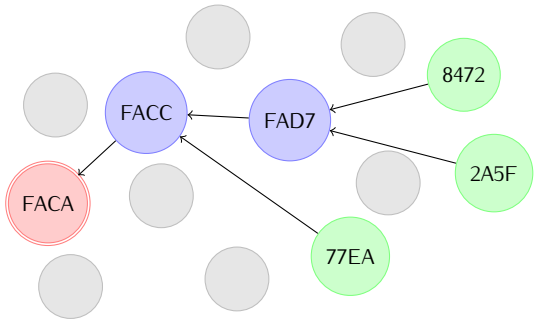
SCRIBE omogoča oddajanje skupini prejemnikov na učinkovit način, tako da informacije o članstvu skupin in zgradbi drevesne strukture, ki je potrebna za učinkovito oddajanje, porazdeli med vsa vozlišča v omrežju. Vsako vozlišče lahko ustvari skupino, druga vozlišča se lahko skupini pridružijo in vanjo oddajajo sporočila, ki so dostavljena vsem članom skupine. Osnovne operacije, ki jih mora zagotoviti vsak protokol za upravljanje oddajanja skupinam prejemnikov in jih ponuja tudi SCRIBE, so naslednje:

- **Ustvari skupino:** Ustvari novo skupino z danim identifikatorjem, ki je predstavljen v obliki niza znakov nabora ASCII.
- **Pridruži se skupini:** Povzroči, da se lokalno vozlišče pridruži določeni skupini. To pomeni, da bo vozlišče od sedaj naprej prejemalo vsa sporočila, ki so poslana v skupino.
- **Zapusti skupino:** Prekliče članstvo v skupini.
- **Pošlji sporočilo:** Pošlje sporočilo vsem članom določene skupine.

Vozlišča prekrivnega omrežja so zadolžena za ustrezno posredovanje kontrolnega prometa za upravljanje s skupinami in za dostavo konkretnih sporočil. Vsaki skupini pripada unikatni identifikator, ki se zgenerira iz uporabniškega niza znakov ASCII s pomočjo zgoščevalne funkcije SHA-1. Na tak način dobimo identifikator v prostoru ključev prekrivnega omrežja. Vozlišče, ki je po razdalji XOR najbližje ključu skupine, proglasimo za točko srečanja (rendezvous point). Le-ta predstavlja koren porazdeljene drevesne strukture, uporabljene za posredovanje sporočil.

Upravljanje članstva skupin

Ustvarjanje skupin poteka z usmerjanjem ustrezno formatiranega sporočila proti korenu skupine. To vozlišče doseže z navadnim posredovanjem sporočila na dani identifikator skupine, usmerjevalni protokoli prekrivnega omrežja Kademlia pa zagotovijo, da sporočilo prejme vozlišče, ki je po razdalji XOR najbližje identifikatorju. Vozlišče, ki sporočilo prejme, registrira skupino v lokalni podatkovni strukturi in tako postane korensko vozlišče za skupino. Zgoščevalna funkcija zagotavlja enakomerno razpršitev identifikatorjev skupin med vozlišči prekrivnega omrežja kar pomeni, da so tudi korenska vozlišča in poti do njih razpršene po omrežju.



Slika 2.9: Porazdeljena drevesna struktura SCRIBE za skupino FACB. Korensko vozlišče označeno z rdečo, posredniška vozlišča z modro, listi z zeleno.

Omenjena porazdeljena drevesna struktura (slika 2.9) ima koren v točki srečanja, drevo pa se gradi s pomočjo sporočil, ki izkazujejo namero nekega vozlišča, da se pridruži skupini. Drevo se gradi v obratni smeri (angl. reverse path forwarding), torej od listov proti korenu. Vozlišče, ki se želi pridružiti skupini, pošlje proti korenu drevesa (identifikatorju skupine) sporočilo o nameri pridružitve. To sporočilo je enakovredno vlogi sporočila IGMP pri upravljanju s skupinami na nivoju IP. Vsako vozlišče na poti do korena, ki prejme to sporočilo (v toku usmerjevalnega protokola prekrivnega omrežja), postane posrednik (angl. forwarder) za dano skupino. Posredniška vozlišča so lahko tudi sama člani skupin za katere posredujejo promet, ni pa to nujno.

Ko posredniško vozlišče prejme namero o pridružitvi, le-to najprej preveri ali je že del drevesa za dano skupino. V kolikor še ni del drevesa, posreduje sporočilo naprej (glede na razreševanje naslednikov po posvetovanju z lokalnimi k-vedri), sicer ga zavrže. V vsakem primeru pa doda vozlišče, iz katerega je prejelo namero na seznam svojih otrok za dano skupino. Korensko vozlišče vedno doda zadnjega posrednika na seznam svojih otrok.

Zapuščanje skupin poteka podobno, vozlišče najprej lokalno označi, da ni več član določene skupine. V kolikor vozlišče zaradi te operacije nima več nobenih otrok (v primeru, da gre za posredniško vozlišče, je namreč povsem možno, da jih še ima) pošlje svojemu staršu sporočilo, da zapušča skupino. Operacija zapuščanja se tako izvede rekurzivno vse do korenskega vozlišča.

Posredovanje sporočil skupini in obnavljanje drevesa

Posredovanje sporočil poteka tako, da vozlišče najprej pridobi kontaktne podatke korena drevesa za ustrezno skupino. Nato posreduje sporočila korenu, ki jih nato posreduje naprej svojim otrokom v drevesu vse do listov. Takšna

dostava sporočil je nezanesljiva in neurejena. V kolikor bi potrebovali večjo zanesljivost in dostavo sporočil v določenem vrstnem redu, bi morali narediti nekatere popravke protokola.

Vsako vozlišče, ki ni list, periodično pošlje vsem svojim otrokom sporočilo, ki oznanja, da je starševsko vozlišče še živo. Seveda kot implicitna obvestila služijo tudi vsa sporočila, ki so posredovana skupini v običajnem toku dogodkov. V primeru, da otrok v določenem časovnem intervalu ne prejme obvestila, smatra starševsko vozlišče za mrtvo in ponovno posreduje namero o pridružitvi skupini preko standardnega usmerjanja v prekrivnem omrežju Kademlia. Sporočilo bo nato usmerjeno k novemu staršu in drevo bo s tem popravljeno.

Prav tako je potrebna periodična osvežitev v obratni smeri, tako da lahko starši izbrišejo nedosegljive otroke iz svojih tabel, kjer hranijo članstvo skupin. Oba postopka za obnavljanje drevesa sta raztegljiva, pošiljanje sporočil se izvaja zgolj v omejeni lokalni okolici reda velikosti $O(\log n)$ vozlišč.

Protokol je sposoben tudi obnove po izpadu korenskega vozlišča. V ta namen so metapodatki, ki jih hrani korensko vozlišče, replicirani med k najbljižjih vozlišč po metriki XOR (gre za k-okolico korenskega vozlišča). V primeru izpada korena bodo bližnja vozlišča že pripravljena na sprejemanje zahtevkov otrok starega korena in tudi za posredovanje sporočil s strani pošiljateljev (le-ti morajo prej osvežiti kontaktne podatke korenskega vozlišča, kar storijo z uporabo navadnega usmerjanja proti identifikatorju skupine preko prekrivnega omrežja).

2.3.2 Prednosti in slabosti

Prednost implementacije oddajanja skupini prejemnikom na aplikacijskem nivoju SCRIBE je v tem, da lahko izkoristimo raztegljivost porazdeljenega omrežja, ki ga ponuja prekrivno omrežje, in da nam ni treba spreminjati obstoječe usmerjevalne infrasturkture.

Slabosti so performančne narave, usmerjanje preko prekrivnega omrežja pomeni večji razteg in večjo obremenitev povezav (po meritvah SCRIBE v povprečju povezave obremeni za štirikrat bolj kot usmerjanje skupinam neposredno preko usmerjevalnih protokolov kot je PIM [6, stran 7]). To je seveda že omenjena pomanjkljivost prekrivnih omrežij, saj je možno informacije o topologiji IP upoštevati samo do določene mere, ki nam jo omejuje sama struktura prekrivnega omrežja.

Poglavje 3

Varnost v prekrivnih omrežjih

Kot smo videli iz pregleda konceptov in različnih rešitev prekrivnih omrežij v prejšnjem poglavju, nam le-ta dajo preprosto vendar močno rešitev za raztegljivo komunikacijo med aplikacijami. Preden pa lahko kakršnokoli takšno rešitev uporabimo v modernem komunikacijskem sistemu, jo je potrebno preučiti tudi z vidika varnosti. Žal se izkaže, da je nad prekrivnimi omrežji, kot so bila le-ta definirana v poglavju 2, možna vrsta napadov [34, 37, 38], ki omogočajo napadalcu različne stopnje manipulacij z omrežjem. Podobne varnostne težave infrastrukturnih storitev (usmerjanje, razreševanje imen) so nam že vrsto let znane tudi v svetu omrežij IP, rešitve zanje pa se uveljavljajo šele v zadnjem času.

Uporaba prekrivnega omrežja za komunikacijo tako odpira dodatne površine za napade in ravno zaradi tega je potrebno raziskati kakšne vrste napadi so možni, da bomo lahko to uporabili pri načrtovanju ogrodja. Pri tem se je potrebno v naprej zavedati dveh pomembnih trditev:

1. Popolna varnost ne obstaja. V vsakem *dovolj kompleksnem* sistemu bodo obstajale varnostne pomanjkljivosti, ki bodo izvirale iz določenih implicitnih predpostavk v interakciji s sistemom.

2. Izboljšanje varnosti navadno pomeni prikrajšanje performančnih sposobnosti sistema in/ali njegove uporabnosti. Najbolj varen računalnik je tisti, ki ni povezan na nobeno omrežje, je izklopljen in zaklenjen v sefu. Seveda tak računalnik ni pretirano uporaben.

Pri načrtovanju sistemov torej velja, da naj bodo varnostne rešitve čim bolj preproste, razumljive in obvladljive. Za cilj si ne smemo izbrati popolne varnosti ampak varnost, ki je zadovoljiva za dani nabor tarč ter obseg napadalcev.

3.1 Kaj in proti čemu zares varujemo

Prva stvar pri preučevanju kakršnegakoli varnostnega vidika je natančna določitev tarč, profilov potencialnih napadalcev ter identifikacija možnih vektorjev napada. Pri raziskovanju smo si kot cilj postavili predvsem uporabo znotraj posamezne organizacije za povezovanje storitev (možnosti za povezave med organizacijami bodo na kratko predstavljene v razdelku 3.5 na strani 38).

Tarča, ki jo varujemo v našem primeru je komunikacijsko ogrodje samo, torej prekrivno omrežje, ki se uporablja za komunikacijo med aplikacijami. Aplikacije navzven vidijo samo pošiljanje sporočil in se tako zanašajo na ogrodje, da sporočila dostavi pravemu naslovniku, ter da so prejeta sporočila avtenticirana (to pomeni da so res prišla od vozlišča, ki je navedeno v glavi sporočila). V kolikor bi imelo ogrodje tudi podporo za avtorizacijo in porazdeljeno upravljanje zaupanja, kar za trenutno verzijo ne drži, bi morali zagotoviti tudi varnost podsistema ter protokolov, ki bi skrbeli za avtorizacijo dovoljenj na nivoju aplikacij. Napadalci na katere se osredotočamo v tem poglavju so naslednji:

- **Zunanji napadalci:** Sem spadajo vsi napadalci, ki nimajo neposrednega dostopa do veljavnega vozlišča, torej napadalci, ki lahko prestrezajo in manipulirajo promet na nivoju IP. Na tem mestu je potrebno zagotoviti varnost sej med posameznimi vozlišči, ter omogočiti povezovanje zgolj med *veljavnimi* vozlišči.

- **Kompromitirana vozlišča:** Ker bodo vedno obstajale napake v operacijskih sistemih, strežniških aplikacijah in nenazadnje tudi v komunikacijskih ogrodjih, obstaja verjetnost, da bodo posamezna *veljavna* vozlišča kompromitirana. V tem primeru gre za vozlišča, ki so pod nadzorom napadalca, ki ima na tem mestu možnost, da prestreza in manipulira promet na nivoju prekrivnega omrežja. Zaščita, ki se uporablja proti zunanjim napadalcem je na tem mestu popolnoma brez vrednosti.

- **Notranji napdalci:** Poleg kompromitiranih vozlišč gre tukaj tudi za morebitne uporabnike oz. aplikacije, ki imajo legitimen dostop do vozlišč, vendar bi želeli priti mimo sistema za avtorizacijo. Ker v tej verziji ogrodja še nimamo skupnega porazdeljenega sistema za podrobno upravljanje zaupanja (tj. sistema za avtorizacijo), varovanja proti tej vrsti napadov ne izvajamo.

V nadaljevanju bomo preučili različne vrste napadov, ki izvirajo iz zgornjih modelov napadalcev. Pri vsakem napadu bomo podali tudi teoretične rešitve

zanje, ki so uporabne za razvoj modernega komunikacijskega ogrodja, ki temelji na prekrivnih omrežjih. Na koncu bomo na hitro osvetlili še dva segmenta na področju varnosti in zaupanja, ki sta dobra kandidata za nadaljnje raziskave.

3.2 Napadi na seje med vozlišči

Vozlišča med seboj komunicirajo s protokolom TCP. To pomeni, da so na tem mestu proti zunanjim napadalcem ranljiva, v kolikor lahko napadalci prestrezajo ali celo manipulirajo s prometom med vozlišči. Možnih je več napadov:

- **Prestrezanje prometa med vozlišči:** Napadalec lahko poskuša prestreči promet med vozlišči, da bi pridobil zaupne informacije iz omrežja. Za tak napad mora napadalec kompromitirati sistem oz. omrežje, ki stoji na poti med posameznimi vozlišči v topologiji IP. Ko je enkrat to storil, ima, v kolikor komunikacija ni ustrezno zaščitena, možnost prebiranja vseh sporočil, ki si jih izmenjujeta vozlišči. Rešitev za to varnostno pomanjkljivost je enkapsulacija vse neposredne komunikacije med vozlišči v standardizirani protokol TLS [10], ki implementira kriptografske algoritme ter protokole za varovanje in avtentikacijo seje.

- **Spreminjanje prometa med vozlišči:** Podobno kot v prejšnji točki ima napadalec v primeru kompromitiranja nekega vmesnega sistema možnost spreminjanja prometa. To lahko izrabi za napade moža v sredini (angl. man-in-the-middle attack), kjer napadalec poskuša izrabiti nezmožnost vzpostavitve zaupanja prek nezavarovanega komunikacijskega kanala med vozlišči. V kolikor mu ta napad uspe, lahko z vsakim izmed vozlišč izpogaja svoj nabor kriptografskih parametrov seje in tako prestreza oz. manipulira tudi notranji promet prekrivnega vozlišča brez vednosti udeleženih vozlišč. Za rešitev tega problema potrebujemo način avtentikacije vozlišč, ki so del veljavnega omrežja. Ena izmed možnih rešitev tega problema je zaupanje določenemu javnemu ključu, ki je v naprej znan vsem vozliščem, ki so del omrežja. Več o tem v razdelku 3.3.1.

- **Zavračanje prometa med vozlišči:** V tem primeru gre v resnici za napad DoS (angl. denial of service), kjer želi napadalec onemogočiti komunikacijo med vozlišči. Proti takim napadom do določene mere varuje že sam koncept prekrivnega omrežja, saj je le-to zelo prilagodljivo in lahko sporočila preusmeri preko drugih vozlišč. Seveda to ni uporabno v kolikor napadalec nadzoruje vse komunikacijske poti danega vozlišča - v tem primeru rešitve ni.

- **Poskus vstopa v omrežje kot veljavno vozlišče:** Napadalec lahko uporabi svojo implementacijo protokolov prekrivnega omrežja za poskus vstopa vanj. Za preprečevanje takega napada potrebujemo najprej koncept in način identifikacije *veljavnega vozlišča*. Možna rešitev tega problema je uporaba kriptografsko podpisanih certifikatov X.509, katere izdaja neka centralna avtoriteta, ki ji vozlišča zaupajo. Ti certifikati se uporabljajo za avtentikacijo v teku protokola TLS (glej prvo točko).

Pred napadi zunanjih napadalcev se je torej relativno preprosto ubraniti v kolikor predpostavimo obstoj centralne avtoritete (javnega ključa), ki ji zaupajo vsa vozlišča. Podobno velja tudi za naslednjo vrsto (notranjih) napadov, imenovano napadi *Sybil*, kjer obstoj centralne avtoritete bolj natančno analiziramo.

3.3 Napadi *Sybil*

Druga vrsta napadov na omrežje komunikacijskega ogrodja je mogoča zaradi same zgradbe prekrivnega omrežja, ki ga uporabljamo. Vsa prekrivna omrežja, ki temeljijo na strukturiranem principu so ranljiva za takoimenovani napad *Sybil* [38]. Tak napad na omrežje je mogoč, v kolikor lahko napadalec pridobi veliko količino identifikatorjev vozlišč ali pa si lahko izbira poljubne identifikatorje. V tem primeru se lahko napadalec vstavi med določene poti v omrežju (tiste, ki gredo čez identifikatorje, ki jih on nadzoruje) in na tak način mu je omogočeno spremljanje, manipulacija in zavračanje prometa na tej poti. To je mogoče zato, ker celotno usmerjanje v strukturiranih prekrivnih omrežjih temelji na strukturi prostora identifikatorjev.

3.3.1 Centralna avtoriteta za izdajanje identifikatorjev

Možno rešitev tega problema predstavlja centralna avtoriteta, ki izdaja identifikatorje vozlišč. Identifikatorji so dodeljeni javnim ključem vozlišč v obliki kriptografsko podpisanih certifikatov X.509, ki jih vozlišča uporabljajo tudi za vzpostavljanje varnih sej preko protokola TLS kar smo omenili že pri obravnavi zunanjih napadalcev. Centralna avtoriteta torej lahko zagotavlja naključno porazdelitev identifikatorjev, saj mora vsako vozlišče svoj identifikator najprej pridobiti od nje. Možna rešitev je centralna avtoriteta kot sistem, ki ni priključen na omrežje. To pomeni, da morajo vozlišča svoje certifikate pridobiti v naprej preko ločenega kanala. To je smiselno za zaprte organizacije, vendar zahteva postavitev svoje infrastrukture javnih ključev (angl. public key infrastructure, PKI).

Takšna centralna avtoriteta sicer rešuje omenjeni problem, vendar postavlja dodatne zahteve na stran uporabnika. Varno upravljanje PKI namreč ni trivialno opravilo. Prav tako se tukaj pojavlja vprašanje varnega prenosa javnega ključa korenskega certifikata (angl. certificate authority key rollover) na nov ključ. Možnost te operacije je absolutno nujno potrebna za možnost varnega delovanja v primeru kompromitiranih ključev ali drugih varnostnih pomanjkljivosti v kriptografskih funkcijah. Porazdeljenost sistema seveda takšne operacije dodatno oteži, saj morajo prenos izvesti vsa vozlišča, v vmesnem času pa morajo delovati vsi ključi. Za rešitev problema bi bilo smiselno raziskovati v smeri več hkrati veljavnih javnih ključev, ki bi skupaj predstavljali centralno avtoriteto in omogočali prenos avtoritete z glasovanjem. Podobno je tudi vprašanje porazdeljenega seznama preklicanih certifikatov (angl. certificatate revocation list, CRL), saj mora obstajati način kako v omrežju preklicati veljavne certifikate. To sta vsekakor težavi, ki ju bo potrebno rešiti z nadaljnjim delom, preden se lahko komunikacijsko ogrodje uporablja v resnih sistemih.

3.3.2 Družabna omrežja (mreža zaupanja)

Obstajajo tudi pristopi, ki temeljijo na uporabi obstoječih družabnih omrežij za preprečevanje napadov *Sybil*, npr. Lesniewski-Laas in sodelavci [23] uporabljajo mrežo zaupanja (angl. web of trust) med posameznimi vozlišči za izboljšanje odpornosti proti tej vrsti napadov. Težava je v tem, da to zahteva vzdrževanje relacij zaupanja med vozlišči (ali pa že vzpostavljeno in v naprej znano zaupanje, kar pa ni pretirano uporabno v konkretnih implementacijah) in periodično ponovno gradnjo usmerjevalnih tabel, kar predstavlja morebitni performančni problem.

3.4 Napadi *eclipse*

Kljub temu, da smo s certificiranimi identifikatorji vozlišč močno omejili možnost napadov, ne smemo spregledati še ene vrste napadov, ki lahko povzroči težave v omrežju tudi v kolikor napadalec nadzoruje manjše število vozlišč [34]. Ker možnost kompromitiranih vozlišč vedno obstaja, je to povsem možen vektor napada na prekrivno omrežje. Pri napadu gre za to, da napadalcu ni potrebno slediti protokolu za upravljanje koherence strukturiranega omrežja, temveč lahko v usmerjevalne tabele vstavlja poljubne kontaktne podatke. To lahko omogoči napadalcu, da tudi z majhnim številom vozlišč nadzoruje veliko število vnosov v usmerjevalnih tabelah in posledično veliko komunikacijskih poti v prekrivnem omrežju.

Za ilustracijo si poglejmo primer napada na osnovno omrežje Kademlia. Usmerjevalna tabela je v njem sestavljena iz k-veder, ki so organizirana v dvojiško drevo, vsak vnos v posameznem vedru pa vsebuje kontaktne podatke. Vozlišča svoje usmerjevalne tabele posodabljajo takoj, ko spoznajo kakšno drugo vozlišče. Spoznavanje vozlišč poteka z vzpostavljanjem povezav z znanimi vozlišči pri operaciji osveževanja k-veder. Pri tem se torej vozlišče popolnoma zanaša na druga vozlišča, da mu vrnejo optimalne kontaktne podatke za zapolnitev svojih k-veder - to pa z varnostnega vidika ustvarja implicitno zaupanje med vozlišči. Vozliščem pod nadzorom napadalca seveda ni treba vračati kontaktov, ki bi bili optimalni za usmerjanje, še več, popolnoma mogoče je, da takšna kompromitirana vozlišča vračajo kontaktne podatke zgolj za druga kompromitirana vozlišča. Na tak način lahko napadalec počasi dodaja kompromitirana vozlišča v usmerjevalne tabele drugih, ki se lahko med nekompromitiranimi vozlišči celo razširjajo zaradi izmenjave podatkov med vozlišči pri različnih operacijah prekrivnega omrežja.

Napadalec sicer nima popolnoma prostih rok pri zapolnjevanju posameznih k-veder, saj morajo identifikatorji ustrezati nekaterim omejitvam, ki izvirajo iz narave strukturiranosti prekrivnega omrežja. K-vedra na višjih nivojih dvojiškega drevesa ponujajo več fleksibilnosti, saj se mora za sodelovanje v k-vedru ujemati manj števk z leve kot pri k-vedrih v nižjih nivojih. Seveda je vse odvisno od tega kakšno mero fleksibilnosti dovoljujemo v posameznih k-vedrih, prva rešitev v boju proti napadom *eclipse* tako naravno izvira iz te opazke in sicer gre za striktnejšo omejitev usmerjevalnih tabel.

3.4.1 Omejene usmerjevalne tabele

Najbolj osnovna rešitev je dodatna omejitev usmerjevalnih tabel oz. vpeljava ločenih omejenih usmerjevalnih tabel (angl. constrained routing tables) [7, 32]. Kot že omenjeno, omogočajo usmerjevalne tabele v omrežju Kademlia določeno mero fleksibilnosti pri zapolnitvi visokoležečih k-veder. Ta fleksibilnost se ponavadi uporabi za izbiro vozlišč, ki so po topologiji IP čim bližje, saj to pomeni optimalnejše usmerjanje (nižji razteg in RTT na nivoju IP). Prav ta fleksibilnost pa omogoča tudi manipulacijo tabel, zato je predlog, da se natančno specificira kateri vnosi se lahko nahajajo v posameznem k-vedru. V primeru omrežja Kademlia bi lahko npr. zahtevali, da se v posameznem vedru lahko nahajajo zgolj vozlišča, ki so med vsemi kandidati najbližje lokalnemu po razdalji XOR. Ker je naključna porazdelitev identifikatorjev zagotovljena s strani centralne avtoritete, to pomeni, da takšna omejitev prepreči vstavitev kontaktov napadalca v večje število vozlišč.

V izogib možni izgubi sporočil pri usmerjanju čez morebitna kompromitirana vozlišča pa bi lahko sporočilo poslali čez več različnih poti (torej preko različnih začetnih vozlišč). Seveda ima ta rešitev (omejene usmerjevalne tabele in redundantno pošiljanje) tudi slabo stran in sicer lahko povzroči precejšnji padec performans zaradi nezmožnosti optimizacij tabel in dodatnega prometa.

3.4.2 Anonimna revizija stopnje vozlišča

Pri drugi rešitvi [31] je ključno spoznanje v tem, da je med napadom *eclipse* stopnja zlonamernih vozlišč višja od povprečne stopnje vozlišč v prekrivnem omrežju. Stopnja vozlišča v grafu (in v prekrivnem omrežju) pomeni število povezav, ki jih ima vzpostavljeno vozlišče z drugimi vozlišči. Torej, vozlišča, ki imajo vzpostavljena nadpovprečno veliko povezav so sumljiva. Iz tega sledi, da je en možen način obrambe proti tej vrsti napada omejitev stopnje navzgor, torej da vozlišča izbirajo za zapolnitev vnosov v usmerjevalnih tabelah samo tista sosednja vozlišča z dovolj nizko stopnjo.

To seveda pomeni, da potrebujemo mehanizem, s katerim lahko vsako vozlišče ugotovi, kakšno stopnjo ima neko sosednje vozlišče. V ta namen avtorji predlagajo porazdeljen algoritem za anonimno revizijo stopnje nekega vozlišča. Vsako vozlišče x hrani seznam vseh svojih sosednjih vozlišč in x periodično izzove vsako vozlišče na tem seznamu, da naj mu pošlje svoj seznam sosedov. V kolikor vrnjeni seznam ne vsebuje vozlišča x ali pa je število vnosov večje od neke meje, x prekine povezavo z omenjenim vozliščem (ga izbriše iz ustreznega k-vedra). Da bi ta postopek resnično deloval, mora biti omenjeni izziv anonimen, saj lahko v nasprotnem primeru napadalec vedno ustrezno priredi svoj seznam, tako da vključuje vozlišče od katerega je dobilo poizvedbo.

Za usmerjanje izzivov zato vozlišča uporabljajo druga vozlišča kot posrednike, ki zakrijejo identiteto pošiljatelja izziva. Recimo, da vozlišče x izbere pri reviziji stopnje vozlišča z neko drugo vozlišče y za anonimizacijo sporočil. Pri tem so možni naslednji štirje scenariji:

1. y **je pošten,** z **je pošten:** Revizija uspe, saj z lahko brez težav odgovori na izziv.

2. y **je pošten,** z **je zlonameren:** Ker vozlišče z ne more vedeti, kdo je zahteval revizijo, lahko ali ne odgovori na izziv ali pa poskusi srečo z vrnitvijo dela seznama, ki je znotraj omejitev (vendar lahko ne vsebuje vozlišča x).

3. y **je zlonameren,** z **je pošten:** V tem primeru lahko y izziva ne posreduje z in ga na tak način lažno obtoži.

4. y **je zlonameren,** z **je zlonameren:** Vozlišči y in z lahko sodelujeta, tako da y razkrije identiteto pošiljatelja, kar omogoča da z uspešno odgovori na izziv. V tem primeru x ne more vedeti, da je karkoli narobe.

Zaradi možnosti zlonamernih posrednikov je pomembno, da x vozlišča ne izključi že po eni neuspešni reviziji, ampak mora to poskusiti večkrat čez različna posredniška vozlišča ob različnih, naključno izbranih, časih (da je otežena ugotovitev identitete s korelacijo sporočil za odkrivanje posrednikov). Za izbiro posredniških vozlišč avtorji protokola priporočajo naključno izbiro med k vozlišči, ki so po razdalji XOR najbližja identifikatorju $id = H(z)$, kjer $H(z)$ predstavlja kriptografsko zgoščevalno funkcijo, ki preslika identifikator vozlišča z v nek drug identifikator prekrivnega omrežja. Za usmerjanje revizijskih izzivov in sporočil za izbiro anonimizacijskih posrednikov je potrebno še vedno uporabljati prej omenjene *omejene usmerjevalne tabele*, kar pomeni da morajo vozlišča vzdrževati dodatno stanje.

Težava s to rešitvijo je v tem, da je učinkovita zgolj, ko je omejena stopnja vozlišča majhna, kar pa ima za posledico daljše čase pri usmerjanju sporočil, tudi ko ni napadov na omrežje [37, stran 25]. To je sicer le ena od možnih rešitev [37], vendar so ostale predlagane rešitve še bolj pomanjkljive oz. problematične pri implementaciji v realnem sistemu. Zaradi tega je torej jasno, da je varovanje prekrivnih omrežij s porazdeljenim zaupanjem še ne rešen problem, ki ga bo potrebno v prihodnosti bolje raziskati.

3.5 Globalno omrežje avtonomnih sistemov

Avtonomni sistem kot ga definiramo za potrebe tega razdelka je entiteta (organizacija, posameznik, delfin), ki ima jasno določeno notranjo politiko zaupanja (oz. model zaupanja) in je v izvajanju te politike neodvisna od drugih avtonomnih sistemov. Med avtonomnimi sistemi lahko obstajajo relacije zaupanja, prav tako pa lahko ti sistemi med seboj interaktirajo in si izmenjujejo sporočila.

Takoj, ko razširimo koncept uporabe ogrodja na omrežje v katerem sodeluje več avtonomnih sistemov (kar je tudi končni cilj, saj želimo, da bi bilo komunikacijsko ogrodje zares univerzalno uporabno) naletimo na nekaj dodatnih varnostnih izzivov:

- **Upravljanje zaupanja postane kritično:** Avtorizacija in upravljanje zaupanja je kritično za dostop do storitev med avtonomnimi sistemi.

- **Varnost usmerjanja postane kritična:** Kompromitiranje vozlišč enega avtonomnega sistema ter izvajanje napadov *Sybil* in *eclipse* znotraj njega ne sme vplivati na usmerjanje znotraj drugih avtonomnih sistemov.

- **Lokalnost poti postane zelo pomembna:** Sporočila morajo ostati znotraj avtonomnega sistema v kolikor je to mogoče (torej v kolikor sta izvor in cilj v istem avtonomnem sistemu).

Osnovna prekrivna omrežja so v nasprotju z zadnjima dvema točkama, saj predpostavljajo, da so vsa vozlišča v omrežju homogena (enaka po svojih lastnostih, varnostnih zahtevah in zaupanju). Za celovito rešitev varne komunikacije med avtonomnimi sistemi pa je potrebno zagotoviti vsaj vse zgornje tri lastnosti. V nadaljevanju si bomo najprej pobližje ogledali koncept avtonomnih sistemov, nato pa razširili osnovna prekrivna omrežja s hierarhično strukturo, ki bo bolje zagotavljala zgoraj omenjeni lastnosti.

3.5.1 Avtonomni sistemi zaupanja in mreža zaupanja

Kot smo nakazali že uvodu, kjer smo si ogledali varnostne zahteve s ptičje perspektive, je centralizirana oblika zaupanja (glej tudi razdelek 3.3.1) za velike in močno porazdeljene sisteme neprimerna. Težava z zaupanjem je namreč dvojna, po eni strani težko zaupamo zasebnim podjetjem ali državnim institucijam, ker bi bile lahko le-te nameroma kompromitirane s strani obveščevalnih služb, po drugi strani pa zato, ker so takšne centralizirane shrambe zaupanja zelo mikavna tarča za napade (ki so popolnoma realna grožnja, kar se je pokazalo ob nedavnem napadu na overovitelja SSL, Comodo CA [18]). Na podobno težavo s centraliziranim zaupanjem naleti tudi razširitev globalnega imenika DNS s protokoli DNSSEC [3].

Drugačno pot uberejo porazdeljeni sistemi zaupanja [11, 35], ki poskušajo modelirati zaupanje med neodvisnimi entitetami v obliki grafa (mreže) zaupanja, katera, če opis karseda poenostavimo, ponazarja koliko si posamezne entitete zaupajo in koliko zaupajo entitetam o ocenjevanju zaupanja drugih entitet. Podoben pristop k zaupanju uporablja tudi ogrodje UIA [14], kjer se relacije zaupanja vzpostavljajo med napravami in družbenimi osebki.

Za komunikacijsko ogrodje je naše izhodišče za nadaljnje delo porazdeljeno upravljanje zaupanja med avtonomnimi sistemi. Takšno upravljanje bi omogočalo avtonomnim sistemom, da sami urejajo zaupanje znotraj sebe in da hkrati lahko vzpostavljajo relacije zaupanja z drugimi avtonomnimi sistemi.

3.5.2 Hierarhična prekrivna omrežja

Kljub modelu porazdeljenega zaupanja nastane še vedno težava pri povezovanju prekrivnih omrežij avtonomnih sistemov. Kot že omenjeno, le-ta namreč predpostavljajo homogenost v zaupanju, kar v osnovi onemogoča avtonomijo. V kolikor bi uporabili ploščato (angl. flat) prekrivno omrežje za postavitev takšne infrastrukture, bi imeli hude težave z ločitvijo vplivov med posameznimi sistemi. Ker so identifikatorji v ploščatih omrežjih popolnoma naključno razpršeni po prostoru, lahko usmerjanje sporočil prečka več različnih organizacij kljub temu, da želimo komunicirati med dvema vozliščema znotraj iste organizacije. Prav tako vsi napadi na en avtonomni sistem vplivajo tudi na vse ostale avtonomne sisteme, s katerimi ima organizacija v danem trenutku vzpostavljene relacije zaupanja.

V ta namen je potrebno razviti drugačno porazdeljeno strukturo prekrivnega omrežja za usmerjanje. Na srečo ta težava ni povsem nova in zato obstaja v literaturi nekaj različnih pristopov [4, 16, 25]. V osnovi se pristopi delijo na dva koncepta in sicer na *horizontalno-hierarhična* prekrivna omrežja ter na *vertikalno-hierarhična* prekrivna omrežja.

Vertikalno-hierarhična

Vertikalno-hierarhična prekrivna omrežja so v resnici sestavljena iz več neodvisnih obročev DHT, ki lahko implementirajo tudi vsak svoj protokol za usmerjanje. Za izmenjavo sporočil med njimi skrbijo prehodi (angl. gateways), ki so v resnici vozlišča, prisotna v več obročih hkrati. Eden izmed primerov takšnega omrežja [25] za usmerjanje sporočil med obroči uporablja infrastrukturo za oddajanje skupinam prejemnikov SCRIBE, ki smo jo podrobneje opisali v razdelku 2.3.1.

Naši avtonomni sistemi zaupanja se v tej obliki neposredno preslikajo v ločene obroče DHT. To pomeni, da vsak avtonomni sistem za svoj obroč vzdržuje usmerjevalne tabele, ki ne vsebujejo vnosov za vozlišča drugih avtonomnih sistemov, kar zagotavlja resnično avtonomijo pri usmerjanju. Za usmerjanje sporočil med avtonomnimi sistemi vertikalno-hierarhična organizacija predpostavlja obstoj korenskega obroča, ki hrani informacije o tem, kje se nahajajo posamezni identifikatorji. Vozlišča, ki služijo kot prehodi med avtonomnimi sistemi na globalnem nivoju se morajo torej poleg svojega internega obroča DHT pridružiti tudi korenskemu obroču. Nižjenivojskim obročem so dodeljeni unikatni identifikatorji, tako da sta za uspešno usmerjanje sporočil potrebna dva podatka:

1. **Identifikator obroča:** Omogoča usmerjanje na nivoju obročev, tako da

sporočilo prispe do enega izmed prehodov za dani obroč oz. avtonomni sistem. V svetu IP bi bili identifikatorji obroča analogni številkam avtonomnih sistemov (angl. autonomous system number, ASN).

2. **Identifikator v prostoru ključev:** Določa, kje znotraj ciljnega obroča se nahaja dano vozlišče oz. ključ. Gre za standardno vrednost v obliki rezultata zgoščevalne funkcije SHA-1.

Kot že omenjeno, se za usmerjanje med obroči uporablja protokol SCRIBE. Vsi prehodi za obroč `77EA` se v korenskem obroču pridružijo skupini z identifikatorjem `77EA` (oz. ustrezno transformiranim identifikatorjem). Podobno se prehodi v notranjem obroču pridružijo skupini z identifikatorjem `0000`, ki predstavlja korenski obroč.

Ko želi nato neko vozlišče posredovati sporočilo v znan obroč, obstajata dva scenarija:

- **Vozlišče je prehod:** V tem primeru je to vozlišče del korenskega obroča in lahko torej pošilja sporočila drugim prehodom preko skupin SCRIBE, ki nosijo identifikator ciljnega obroča. Za dostavo sporočila ga mora torej zgolj posredovati v ustrezno skupino in ciljni prehodi ga bodo dostavili ustreznemu vozlišču znotraj ciljnega avtonomnega sistema.

- **Vozlišče ni prehod:** To pomeni, da vozlišče nima povezave s korenskim obročem in da mora svoje sporočilo v drug obroč posredovati preko lokalnega prehoda. Sporočilo zato posreduje skupini `0000`, ki vključuje vse prehode znotraj avtonomnega sistema. Ti prehodi nato posredujejo sporočilo preko ustrezne ciljne skupine v korenskem obroču kot v prvem primeru.

Seveda identifikatorji obročev niso vedno v naprej znani, zato omenjeni sistem predpostavlja obstoj porazdeljenega preslikovalnega sistema v korenskem obroču, ki preslika globalne identifikatorje v identifikatorje obročev. Pred usmerjanjem je v tem primeru potrebno izvesti še razreševanje identifikatorjev. Opisani postopki veljajo za dvonivojsko strukturo, vendar so enostavno razširljivi tudi na več nivojev [25, stran 3].

Takšno vertikalno-hierarhično prekrivno omrežje omogoča večjo avtonomijo in nadzor lokalnosti. Sporočila med vozlišči znotraj avtonomnega sistema ostanejo v njem, napadi znotraj lokalnih obročev ne morejo vplivati na druge avtonomne sisteme. Težavo pa predstavlja ravno predpostavka o korenskem obroču, ki je namenjen izmenjavi sporočil med organizacijami in hranjenju preslikav med globalnimi identifikatorji in identifikatorji obročev (oz. avtonomnih sistemov).

Ta obroč je spet lahko tarča za napade iz katerekoli organizacije, ki v njem sodeluje. Možna rešitev bi bila, da se na korenskem nivoju za usmerjanje ne uporablja prekrivnega omrežja, ampak da organizacije med sabo komunicirajo z nekim drugačnim protokolom. Ta rešitev pa ima takoj težave z raztegljivostjo, saj prekrivna omrežja rešujejo ravno problem raztegljivosti pri velikem številu vozlišč.

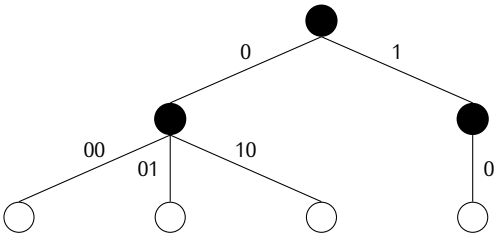
Horizontalno-hierarhična

Drugačen pristop uberejo rešitve s horizontalno hierarhijo [4, 16]. Namesto neodvisnih podobročev, vsak s svojo množico identifikatorjev vozlišč, se pri horizontalno-hierarhičnih prekrivnih omrežjih uporablja en sam prostor identifikatorjev. Identifikator vozlišča je torej sestavljen iz dveh delov, prvi del predstavlja identifikator vozlišča znotraj podobroča, recimo mu $nodeId$, drugi del pa predstavlja hierarhični identifikator podobroča, recimo mu $clusterId$. To na prvi pogled izgleda zelo podobno kot v primeru vertikalno-hierarhičnih omrežij, vendar tukaj ne gre za dva ločena identifikatorja - v resnici sta identifikatorja združena:

$$id = nodeId||clusterId$$

Kaj nam taka organizacija prostora identifikatorjev v resnici pomeni? Na tak način strukturirani identifikatorji nam razdelijo celotni prostor identifikatorjev na več disjunktnih podprostorov. Vzemimo za primer identifikator podobroča `7EA` - ker je identifikator podobroča na koncu celotnega identifikatorja vozlišč, bo med dvema zaporednima vozliščema v tem podobroču v resnici razdalja 2026 (desetiška vrednost identifikatorja podobroča). Ker tak način organizacije v resnici segmentira prvotni prostor identifikatorjev, se tako ohranijo performančne lastnosti ploščatih (nehierarhičnih) prekrivnih omrežij, ki se zanašajo na enakomerno razpršenost identifikatorjev preko celotnega prostora [4, stran 3] - razpršitev $nodeId$ bo avtomatsko pomenila tudi razpršitev id.

Omenili smo, da so identifikatorji podobročev (če za trenutek pogledamo nazaj k avtonomnim sistemom zaupanja, bi vsakemu takemu avtonomnemu sistemu dodelili en podobroč) hierarhični. To pomeni, da znotraj identifikatorjev obstaja hierarhična struktura poljubne globine, ki je za boljšo predstavo prikazana na sliki 3.1. V resnici bi za implementacijo naših avtonomnih sistemov zadostovala že dvonivojska struktura. Najbolj pomembna operacija nad to hierarhično strukturo je združevanje podobročev (angl. interconnection of child DHT clusters). Združevanje pomeni vzpostavljanje povezav (oz. zapolnitev vnosov v usmerjevalnih tabelah) med vozlišči na tak način, da lahko vozlišča vseh



Slika 3.1: Hierarhična razdelitev podobročev. Polni obroči so *združeni podobroči*, ki vsebujejo vsa vozlišča obročev otrok. Bela vozlišča predstavljajo prostor ključev, ki ga definira $nodeId$ (brez $clusterId$). Dvojiška števila na povezavah predstavljajo del identifikatorja $clusterId$, ki se hierarhično sestavlja. Identifikatorji vozlišč v drugem belem podobroču z leve bi se torej vsi končali z `010`.

združenih podobročev med sabo komunicirajo in da hkrati ohranijo logaritemsko razteglјivost prekrivnega omrežja.

Združevanje poteka rekurzivno vse do korena, kar pomeni da lahko v končni fazi med sabo komunicirajo prav vsa vozlišča znotraj tega hierarhično organiziranega prekrivnega omrežja. Za vzpostavljanje povezav in usmerjanje med podobroči (oz. v našem primeru avtonomnimi sistemi) to hierarhično prekrivno omrežje uporablja metriko XOR [4, stran 5], takšna organizacija pa je podobna že omenjenemu omrežju Kademlia, le da gre tukaj za povezave na nivoju celotnih podobročev in ne posameznih vozlišč.

Poglavje 4

Arhitektura UNISPHERE

Do sedaj smo obdelali vso potrebno teorijo, tako z vidika usmerjanja po strukturiranih prekrivnih omrežij, kot z vidika varnostne analize. Že v uvodu smo nakazali nekatere ključne točke, ki v resnici predstavljajo glavne zahteve modernih komunikacijskih ogrodij za gradnjo porazdeljenih sistemov. Če jih ponovno na hitro preletimo, so to naslednje zahteve:

- Ogrodje mora biti enostavno za uporabo razvijalcem in mora skriti detajle komunikacij v porazdeljenih sistemih. Na ta način se zniža krivulja učenja in ogrodje postane bolj dostopno ter obvladljivo, gradnja aplikacij pa cenejša.

- Zagotovljena mora biti podlinearna raztegljivost, tako da se sistem dobro odziva tudi pri velikem številu vozlišč.

- Vse komponente ogrodja se morajo zavedati varnostnih problematik v porazdeljenih sistemih, ki za komunikacijo uporabljajo obstoječa lokalna in prostrana omrežja. V ta namen morajo implementirati ustrezno stopnjo zaščite, da se morebitne napadalce pri malomarnem početju čim bolj omeji.

V tem poglavju predstavljamo arhitekturo in implementacijo našega komunikacijskega ogrodja UNISPHERE, ki zadosti vsem tem zahtevam. Implementacija je zasnovana kot nabor C++ knjižnic, ki jih lahko razvijalec vključi v svojo aplikacijo in s tem pridobi možnost transparentne komunikacije med vozlišči. Vse podrobnosti komunikacije so abstrahirane, navzven je razvijalcu (uporabniku ogrodja) viden zgolj vmesnik za pošiljanje sporočil.

4.1 Razširitev DHT za usmerjanje sporočil

Kot že omenjeno, so protokoli omenjenih prekrivnih omrežij zasnovani kot porazdeljene zgoščevalne tabele. To pomeni, da so primarno namenjeni shranjevanju in branju vrednosti, ki so identificirane s ključi. V našem ogrodju pa želimo protokole uporabljati tudi za usmerjanje sporočil, za kar je potrebna manjša prilagoditev. Pri njeni zasnovi se opiramo na rešitev omenjeno v opisu arhitekture UIA [14], bolj natančno na protokol, ki ga avtorji poimenujejo IHR (angl. Identity Hash Routing). Le-ta naravno spremeni namembnost prekrivnega omrežja Kademlia, tako da omogoča gradnjo komunikacijskih kanalov med vozlišči preko poljubne topologije na nivoju IP.

4.1.1 Predpostavke protokolov DHT in njihove težave

Težava pri uporabi v poglavju 2 omenjenih prekrivnih omrežij za splošno usmerjanje sporočil sledi iz nekaterih predpostavk o homogenosti vozlišč, ki smo se jih dotaknili že pri analizi varnostnih vidikov v poglavju 3, kjer smo omenili težavo s predpostavko o homogenosti zaupanja med vozlišči.

Tokrat si poglejmo še eno pomembno predpostavko, ki jo zahtevajo vsi osnovni protokoli za vzpostavitev prekrivnih omrežij in ki neposredno omejuje njihovo uporabo. Gre za predpostavko o univerzalni povezljivosti med vozlišči, ki zahteva, da je možna neposredna in simetrična (obojesmerna) komunikacija med katerimkoli parom vozlišč, ki so del istega prekrivnega omrežja. Ta predpostavka v realnih internetnih topologijah naleti na težavo, ki ji pravimo *netranzitivnost povezav* [15]. Predstavljajmo si, da imamo vozlišča A, B in C. Netranzitivnost povezav pomeni, da vozlišče A od vozlišča B pridobi kontaktne podatke vozlišča C, vendar zaradi težav v povezljivosti nima povezave do tega vozlišča kljub temu, da lahko vozlišče B normalno komunicira z vozliščem C. Težava se lahko pojavi zaradi naslednjih faktorjev:

- **Različni naslovni prostori vozlišč:** Vozlišča se lahko nahajajo na različnih naslovnih prostorih na nivoju IP, recimo zaradi uporabe zasebnega naslovnega prostora (kjer so v IPv4 vmes velikokrat prisotni prevajalniki omrežnih naslovov) ali pa zato, ker imajo nekatera vozlišča zgolj naslove IPv4, druga pa zgolj naslove IPv6. Omenjeni prevajalniki omrežnih naslovov težavo še povečajo, saj lahko obstaja nesimetrična povezljivost že ko imamo samo dve vozlišči. V primeru, da preusmeritev vrat (angl. port forwarding) ni ustrezno nastavljena, je brez trikov možna povezava zgolj v eno smer.

- **Filtri in prehodne težave pri usmerjanju na nivoju IP:** Usmerjevalniki na nivoju IP lahko filtrirajo promet med določenimi naslovi, kar onemogoča polno povezljivost. Prav tako se lahko med normalnim delovanjem v internetnem omrežju pojavljajo kratkotrajne težave z dosegljivostjo zaradi izpadov povezav, težav s posodobitvami BGP, ipd. Takšne težave so velikokrat omejene na določene avtonomne sisteme (tukaj z avtonomnimi sistemi ne mislimo tistih iz poglavja 3, temveč mislimo avtonomne sisteme iz protokola BGP), kar povzroči nekonsistenten pogled na povezljivost vozlišč, ki se nahajajo v različnih avtonomnih sistemih.

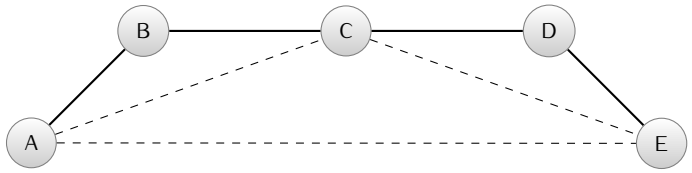
V primeru, da bi za usmerjanje uporabljali katerega izmed *link-state* usmerjevalnih protokolov, tega problema ne bi bilo, saj se lahko tam usmerjevanje popolnoma prilagodi topologiji omrežja IP. V primeru uporabe strukturiranega prekrivnega omrežja pa nam struktura omrežja diktira katere povezave morajo biti vzpostavljene med katerimi vozlišči. Če sta namreč dve vozlišči soseda v prostoru identifikatorjev prekrivnega omrežja obstaja velika verjetnost, da mora biti med njima vzpostavljena povezava - v nasprotnem primeru eno vozlišče lahko sploh ne bo dosegljivo, ker ne bo vidno. Struktura identifikatorjev pa seveda nima nobene korelacije s topologijo na nivoju IP, kar otežuje fleksibilnost. V primeru omrežja Kademlia je možno ta problem omiliti pri *dolgih* (glede na metriko XOR) začetnih skokih, saj je tam za vnos v ustreznem k-vedru na voljo veliko kandidatov in vozlišče lahko prosto izbere tistega z najboljšo povezljivostjo (glede na neko metriko). Težava pa se pojavi pri krajših skokih, kjer smo glede izbire precej omejeni.

Za te primere potrebujemo način, ki nam bo omogočal vzpostavljanje povezav tudi med takšnimi vozlišči, ki zaradi zgoraj omenjenih razlogov morda niso neposredno dosegljiva.

4.1.2 Navidezne povezave in tuneli

Vsekakor imamo opravka s težavo pri zagotavljanju povezljivosti med vozlišči, najboljša rešitev problema pa je dvodelna, torej hkratna aplikacija dveh komplementarnih konceptov:

1. **Tuneliranje povezav:** Prekrivno omrežje lahko uporabimo za odkrivanje vozlišč, ki bi lahko delovala kot posredniki prometa med drugimi vozlišči, ki sicer nimajo neposredne povezave. Čez takšna vozlišča nato vzpostavimo navidezne povezave, ki lahko prečkajo več posredniških vozlišč. Ta rešitev zahteva razširitev protokola na tak način, da le-ta poleg dejanskih



Izvorno usmerjanje

Vozlišče	Vrsta	Pot
B	ip	127.0.0.1:8472
C	tunel	A → B → C
E	tunel	A → B → C → D → E

Rekurzivno tuneliranje

Vozlišče	Vrsta	Pot
B	ip	127.0.0.1:8472
C	tunel	A ⇒ B ⇒ C
E	tunel	A ⇒ C ⇒ E

Slika 4.1: Prikaz dveh načinov usmerjanja preko tunelov, izvorno usmerjanje v usmerjevalnih tabelah uporablja za specifikacije samo povezave na nivoju IP (→), medtem ko rekurzivno tuneliranje uporablja tudi že obstoječe tunele (⇒). Prikazani sta poenostavljeni usmerjevalni tabeli za vozlišče A za oba načina usmerjanja.

povezav na nivoju IP omogoča tudi gradnjo navideznih povezav. Navidezne povezave bi lahko primerjali s tuneli, ki se v svetu IP uporabljajo za vzpostavitev navideznih zasebnih omrežij (angl. Virtual Private Networks, VPNs), le da tukaj ne izvajamo enkapsulacije.

2. **Prečkanje prevajalnikov omrežnih naslovov *(angl. NAT traversal)*:** Kot dodatno optimizacijo lahko uporabimo postopke za prečkanje prevajalnikov omrežnih naslovov, ki v določenih primerih omogočajo vzpostavitev povezav med vozlišči, kjer le-ta zaradi prevajalnikov naslovov sicer ne bi bila mogoča. Tukaj se uporabljajo tehnike kot je npr. vrtanje lukenj (angl. hole punching), ki z uporabo pomožnega vozlišča vzpostavi dvosmerno komunikacijo med dvema vozliščema, ki sta za prevajalniki omrežnih naslovov [14, stran 121].

Za podporo tuneliranu povezav je potrebno obstoječo strukturo prekrivnega omrežja Kademlia malenkost razširiti. Kot že omenjeno v razdelku 2.2.4, se v k-vedrih usmerjevalne tabele nahajajo kontaktni podatki vozlišč. Ti kontaktni podatki so v osnovni različici protokola lahko zgolj pari $(ip, port)$, ki označujejo, kje je vozlišče mogoče kontaktirati neposredno na nivoju IP. To strukturo kontaktnih podatkov sedaj razširimo, tako da lahko vsebuje vrsto različnih načinov komunikacije - eden izmed njih pa so tudi navidezne povezave preko tunelov.

Načini razširitve kontaktnih podatkov

Vsak tunel je sestavljen iz ene ali več povezav in ustvarja navidzeni komunikacijski kanal med vozliščema, ki sta na skrajnih koncih. Tunel med vozliščema A in B lahko torej definiramo kot vektor $T_{AB} = (P_{A1}, P_{12}, P_{23}, ..., P_{nB})$. Element P_{ij} predstavlja povezavo med vozliščema i in j, čez katero poteka del tunela. Promet lahko vstopi samo skozi prvo ali pa skozi zadnjo povezavo in nato poteka v ustrezni smeri. Namenoma smo uporabili definicijo, ki uporablja povezave namesto definicije, ki bi uporabljala vozlišča, saj nam definicija s povezavami daje večjo opisno zmožnost, kot bomo videli v nadaljevanju. Način izvedbe samega tuneliranja za uporabo v komunikacijskem ogrodju ni pretirano pomemben, bistvo je, da za višjenivojski vmesnik tunel T_{AB} izgleda popolnoma enako kot katerakoli neposredna povezava med A in B (v kolikor bi ta obstajala).

V prej omenjenem članku [14] sta opisani dve rešitvi za gradnjo tunelov, ki neposredno vplivata na način hranjenja kontaktnih podatkov v usmerjevalnih tabelah (glej sliko 4.1). Razlika med obema načinoma je v tipu elementov P_{ij}, ki lahko nastopajo v opisu posameznega tunela. Prvi način je *izvorno usmerjanje* (angl. source routing), kjer so lahko elementi, ki opisujejo tunele zgolj druge *neposredne* povezave (torej povezave vzpostavljene neposredno preko nivoja TCP). Pri drugem načinu, ki mu pravimo *rekurzivno tuneliranje* (angl. recursive tunneling), pa so elementi P_{ij} lahko tudi drugi tuneli T_{ij}, kar pomeni da so tuneli definirani rekurzivno.

Končni učinek (komunikacija med vozliščema A in B) je v obeh primerih popolnoma enak, ključna razlika pa je v znanju, ki ga nosi vsako posamezno vozlišče o vzdrževanem tunelu. V primeru izvornega usmerjanja pozna izvorno vozlišče (v primeru s slike 4.1 je to vozlišče A) celotno pot na kateri se vzpostavlja tunel (in ima tako tudi večji nadzor nad njegovim potekom), medtem ko je v primeru rekurzivnega tuneliranja povezava skozi tunel C $\rightarrow$ D $\rightarrow$ E za vozlišče A popolnoma nevidna. Ključna prednost pri večjem nadzoru nad potekom tunelov v primeru izvornega usmerjanja je boljša možnost optimizacije poti [14, stran 82] in zagotavljanje redundančnih tunelov. Seveda ima večji nadzor nad potekom tunelov tudi nekaj slabosti in sicer:

- **Večja prostorska zahtevnost:** Ker morajo usmerjevalne tabele v primeru tunelov hraniti celotno pot, se prostorska zahtevnost le-teh v najslabšem primeru poveča na $O(n \log n)$ (do tega bi prišlo v primeru, da bi bila vozlišča povezana v dolgo verigo, kjer bi bili na nivoju IP dostopni zgolj sosednji dve vozlišči v taki verigi). V realnih topologijah je verjetnost za takšno povečanje zelo majhna, postaviti pa je mogoče tudi omejitve števila posredniških vozlišč pri vzpostavitvi tunelov (to seveda lahko pomeni, da

določena vozlišča ne bodo dosegljiva).

- **Ponovna vzpostavitev v primeru izpada povezav:** V primeru sprememb stanja povezav na omrežju je potrebno tunele, ki uporabljajo izvorno usmerjanje, ponovno zgraditi. Pri rekurzivnem tuneliranju so podrobnosti usmerjanja skrite, kar pomeni da se dejanske poti samodejno prilagajajo razpoložljivosti omrežja.

V internetnih topologijah je daleč najbolj pogost pojav tuneliranje preko nekega dobro povezanega vozlišča - v terminologiji omrežij P2P se za taka vozlišča uporablja izraz supervozlišče (angl. supernode). To pomeni, da bodo zelo verjetno najbolj pogosti tuneli vključevali zgolj tri vozlišča (izvorno, posredniško in ciljno). Zaradi tega smo se pri razvoju UNISPHERE osredotočili na izvorno usmerjanje, vendar je ogrodje razširljivo, tako da obstaja tudi možnost hibridnih razširitev.

Gradnja povezav in združitve particij ob izpadih

Gradnja povezav poteka zelo podobno kot iterativna operacija *najdi vozlišče* v prekrivnem omrežju Kademlia. Ob približevanju ciljnemu vozlišču po številkah si izvorno vozlišče shranjuje kontaktne podatke spoznanih vozlišč. Vsakič do na novo spoznanega vozlišča (naslednjega koraka) poskuša najprej vzpostaviti neposredno povezavo na nivoju IP. V kolikor le-ta ne uspe, poskuša vzpostaviti tunel preko vozlišča, ki je sosed tega na novo spoznanega vozlišča. Povezava do takšnega soseda vedno obstaja, saj smo jo vzpostavili v prejšnjem koraku (preko nivoja IP ali pa preko tunela). V primeru ponovne uporabe tunelov enostavno združimo poti.

Pomemben dodatek, ki izhaja iz sprostitve predpostavk omrežja Kademlia, je tudi algoritem za združevanje omrežnih particij ob izpadih. Osnovna prekrivna omrežja namreč predpostavljajo, da so izpadi vozlišč relativno neodvisni [14, stran 76], to pa v našem primeru ne drži, saj so nekatere povezave vzpostavljene preko tunelov. Izpad takšnih posredniških vozlišč pa lahko prekine celo vrsto povezav in tako lažje particionira omrežje. Tudi v kolikor ne bi imeli tunelov, bi bile takšne masovne razdružitve možne v primeru izpadov povezav med avtonomnimi sistemi ponudnikov internetnih storitev. Zaradi tega potrebujemo način, kako omrežje v teh primerih ponovno združiti, saj takšne razdelitve pri usmerjanju niso zaželjene, ker pomenijo nedosegljivost.

Da bi lahko zaznali kdaj se mora omrežje ponovno združiti, lahko uporabimo kontaktne podatke iz predpomnilnika ali pa nek drugačen način odkrivanja vozlišč (recimo preko razrešitev znanih DNS naslovov ali pa na lokalnih omrežjih

preko protokola Zeroconf, s katerim lahko odkrijemo sosednja vozlišča na topologiji IP). Ko je enkrat povezava med takim parom vozlišč vzpostavljena, je potrebno na obeh straneh izvesti operacijo združitve, ki ustrezno popravi vnose v usmerjevalnih tabelah omrežja.

Prva stopnja algoritma za združevanje je v resnici zelo podobna algoritmu za vstavljanje novih vozlišč v osnovnem protokolu Kademlia, le da se tukaj na vsakem koraku uporablja razširjena metoda vzpostavljanja povezav med vozlišči (ki lahko uporabi tako neposredne povezave IP kot tudi tunele). S pomočjo osveževanja k-veder (za podrobnosti glej razdelek 2.2.4) se napolnijo lokalne usmerjevalne tabele in vzpostavijo ustrezne povezave med vozlišči [14, stran 77].

Druga stopnja pa vsebuje obveščanje ostalih vozlišč o združitvi, tako da lahko tudi le-ta posodobijo svoje usmerjevalne tabele in tako zagotovijo ustrezno povezljivost (na tem mestu lahko opazimo podobnost s prekrivnim omrežjem SkipNet, opisanim v razdelku 2.2.3, ki tudi uporablja algoritem za združevanje, vendar je ta zaradi strukture omrežja bolj zapleten in zahteva sinhronizacijo [19, stran 16]). Ko vozlišče n spozna neko novo vozlišče n_s in lokalno k-vedro v katerega pade n_s ni bilo zapolnjeno pred dodajanjem tega vozlišča, n obvesti svoje sosede o novem vozlišču n_s. Ko sosedje prejmejo takšna obvestila, le-ti prav tako dodajo vozlišče n_s v ustrezno k-vedro, vzpostavijo povezavo z danim vozliščem, izvedejo algoritem združevanja zanj in tako ponovijo postopek obveščanja v kolikor to k-vedro ni bilo zapolnjeno.

Na tem mestu je seveda potrebno paziti, da pri združitvi večjih omrežij ne pride do poplavljanja omrežja z obvestili o združevanju. UNISPHERE zato zahteva predpomnenje že prejetih obvestil (tako da se že izvedene združitve ne ponavljajo) in uporablja časovnike pri njihovem posredovanju, tako da se v primeru presega določenega števila obvestil vsa nadaljnja obvestila zakasnijo in se izvedejo kasneje.

4.2 Hiter pregled izbranih tehnologij

Ogrodje UNISPHERE za svojo implementacijo uporablja vrsto obstoječih programskih rešitev, ki jih v tem razdelku na kratko predstavimo in opišemo. Kljub temu, da so razvite rešitve načeloma prenosljive med platformami, je zaenkrat arhitektura testirana in pripravljena zgolj za odprtokodni operacijski sistem Linux. Prav tako so vse izbrane tehnologije na voljo pod odprtokodnimi licencami, kar omogoča vpogled v njihovo delovanje in morebitne prilagoditve. Pri posamezni tehnologiji opišemo tudi ozadje, ki stoji za njenim osnovnim delovanjem in način uporabe v našem komunikacijskem ogrodju.

4.2.1 C++, Boost in Boost.ASIO

Pri razvoju komunikacijskega ogrodja smo se odločili za programski jezik C++. Gre za moderen objektno-orientiran jezik, ki podpira več različnih načinov programiranja (proceduralni, funkcijski, objektno-orientirani, ...) in bo v kratkem dobil pomembno osvežitev v obliki novega standarda C++11 (določene funkcionalnosti le-tega že uporablja tudi naše ogrodje UNISPHERE). Za nemoten razvoj potrebujemo tudi bogato knjižnico, ki dopolnjuje jeziku priložen STL (angl. Standard Template Library). V našem primeru smo se odločili za odlično knjižnico Boost, ki implementira vrsto uporabnih razširitev na različnih področjih.

Za delo s komunikacijskim skladom IP v naboru Boostovih knjižnic obstaja Boost.ASIO, ki omogoča enostaven razvoj asinhronih dogodkovno orientiranih omrežnih aplikacij. Na tak način je zasnovano tudi naše ogrodje UNISPHERE. Omrežnih aplikacij oz. poljubnih vhodno-izhodnih operacij se je namreč mogoče lotiti na dva osnovna načina, sinhrono in asinhrono.

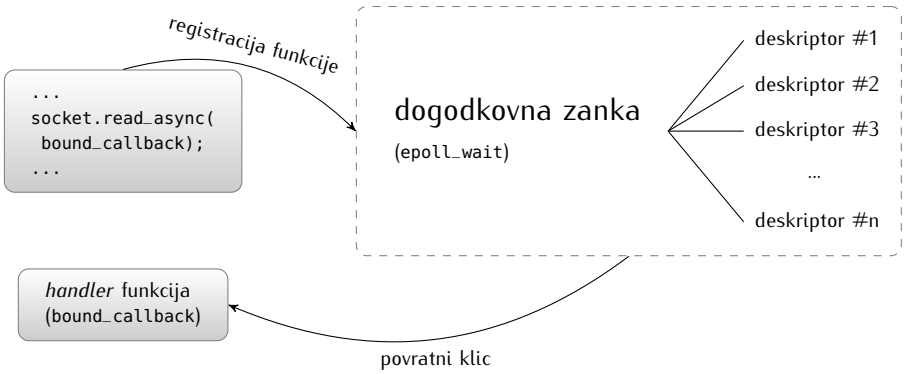
Sinhrono (blokirajoče) omrežno programiranje

Najbolj preprosta oblika programiranja omrežnih aplikacij, kjer vhodno-izhodne operacije izvajamo zaporedno, posamezne funkcije pa ob klicu blokirajo trenutni proces oz. nit je sinhron oz. blokirajoč način. Za primer vzemimo npr. ukaz standardnih POSIX vtičnikov `read`. Ob klicu tega ukaza v blokirajočem načinu bo operacijski sistem prekinil izvajanje naše niti in jo nadaljeval šele takrat, ko bo na povezavi na voljo ustrezna količina podatkov. Ker lahko tako delamo samo z eno povezavo naenkrat (saj je naši niti med čakanjem vzeto izvajanje in torej spi), potrebujemo za več povezav v tem primeru nujno več niti (oz. procesov).

To pa takoj prinese dodatne zahteve po procesorskem času, pomnilniku in v primeru uporabe več niti znotraj istega pomnilniškega prostora dodatne probleme s sinhronizacijo podatkovnih struktur. Takšnemu načinu programiranja vhodno-izhodnih aplikacij se zato v kompleksnejših, modernih in hitrih omrežnih aplikacijah izogibamo.

Asinhrono (dogodkovno) omrežno programiranje

Pri asinhronem načinu pa se operacije obnašajo popolnoma drugače, namesto da bi operacija zaustavila našo nit dokler podatki niso na voljo, se izvajanje programa nadaljuje. To pomeni, da v tem primeru ne moremo *preprosto* uporabljati zaporedja ukazov, saj se ti izvedejo preden lahko nadaljujemo s tokom programa. Ubrati je potrebno drugačno metodo programiranja, takšno z uporabo povratnih klicev (angl. callbacks) - npr. ob klicu metode `read` registriramo



Slika 4.2: Prikaz poteka izvedbe povratnega klica pri asinhronem omrežnem programiranju v psevdokodi. Ob klicu `read_async` se registrira *handler* povratnega klica, katerega izvede dogodkovna zanka kasneje, ko dobi ustrezno obvestilo o dogodku preko `epoll_wait`. Tok programa se po klicu `read_async` nadaljuje brez prekinitve.

metodo za povratni klic, ki se bo izvedla, ko bodo podatki na voljo, oz. ko bo prišlo do napake. Celotna arhitektura mora biti zato pri dogodkovnem načinu organizirana kot nekakšen končni avtomat s prehodi med različnimi stanji, ki so vodeni s strani dogodkov.

Tukaj se seveda pojavi vprašanje, kdo kliče naše registrirane metode. Programi, ki uporabljajo dogodkovni način izvajanja omrežnih operacij vsebujejo osrednjo dogodkovno zanko (angl. event loop). Ta zanka se nahaja v eni izmed niti programa (lahko tudi v več njih) in posluša za dogodke na datotečnih deskriptorjih, ki predstavljajo vtičnike (glej sliko 4.2). Različni operacijski sistemi ponujajo različne vmesnike za učinkovito poslušanje dogodkov za veliko število deskriptorjev, na Linuxu ima to vlogo mehanizem `epoll`. Na popolnoma enak način so implementirani tudi vsi časovniki (angl. timers) v aplikaciji, proži jih torej dogodkovna zanka, ki predstavlja osnovno vstopno točko vsakega dogodkovno orientiranega programa.

Omenjeni mehanizem je preprosti vmesnik (angl. API), ki ga daje na voljo Linux jedro. Omogoča učinkovito (časovna zahtevnost $O(1)$ pri n deskriptorjih) preverjanje ali je na katerem izmed datotečnih deskriptorjev prišlo do *dogodka*. Kaj predstavlja dogodek za posamezni datotečni deskriptor, je odvisno od zadnje operacije, ki je bila izvedena nad deskriptorjem. Lahko gre za branje, pisanje, sprejemanje nove povezave ali pa za izjemno situacijo, kjer je prišlo do napake v delovanju. Podobni mehanizmi so na voljo tudi na drugih operacijskih sistemih z zelo podobnimi vmesniki (npr. `kqueue` na FreeBSD in MacOS X).

Pri uvedbi vzporednosti imamo tukaj na voljo dva načina. Za izvajanje vseh operacij lahko uporabimo eno nit, kjer za preklop med izvajanjem različnih delov programa skrbi zgolj dogodkovna zanka s proženjem povratnih klicev. Drugi način pa je hibridni način, kjer uporabimo več niti, vsako s svojo dogodkovno zanko in razporedimo izvajanje operacij med več procesorjev. V primeru uporabe več niti seveda prav tako naletimo na dodatne komplikacije s sinhronizacijo podatkovnih struktur za katero moramo uporabljati *mutexe*. C++ knjižnica Boost.ASIO podpira oba načina delovanja, ki ju podeduje tudi naše ogrodje.

Za abstrakcijo vseh teh mehanizmov poskrbi že omenjeni Boost.ASIO. Le-ta daje na voljo tudi preprost objektno-orientiran in razširljiv C++ vmesnik za dogodkovno programiranje. Implementira lahek ovoj (angl. wrapper) okrog osnovnih POSIX vtičnikov in jih vključi v svojo implementacijo dogodkovne zanke, ki temelji na `epoll` oz. ustreznih mehanizmih glede na platformo. Prav knjižnica Boost.ASIO se v ogrodju UNISPHERE uporablja kot osrednji dispečer vseh dogodkov ogrodja.

4.2.2 OpenSSL in Botan

Za vzpostavitev varnih povezav v ogrodju uporabljamo knjižnico OpenSSL, ki predstavlja implementacijo protokola TLS (angl. Transport Layer Security) in sicer jo uporabljamo v okviru TLS ovoja, ki se nahaja v Boost.ASIO. Na žalost ima knjižnica OpenSSL precej neprijazen programski vmesnik, zato za ostale kriptografske operacije uporabljamo C++ knjižnico Botan.

Ta knjižnica implementira tako simetrične kot asimetrične šifre, kriptografske zgoščevalne funkcije, pakiranje in nalaganje X.509 certifikatov ipd. V ogrodju je najbolj prisotna znotraj modula *Identity* (več o njem v nadaljevanju), ki implementira vmesnike za dostop do certifikatov in zasebnih ključev vozlišč.

4.2.3 Google Protocol Buffers

Za prenašanje sporočil, ki jih zahtevajo protokoli ogrodja UNISPHERE, je potrebno uporabiti nek serializacijski zapis. Takšna sta npr. DER in BER, ki sta znana iz certifikatov X.509 ali pa zloglasni in prenapihnjeni XML.

V našem ogrodju smo za serializacijski format izbrali Googlove *Protocol Buffers*. Gre za prenosljiv, kompakten in predvsem razširljiv format, kjer podamo opis sporočil v platformno in jezikovno neodvisni obliki, nato pa jo prevajalnik `protoc` prevede v ustrezen jezik. Podprti so vsi pomembni jeziki kot npr. C++, Java, Python, C# in mnogi drugi.

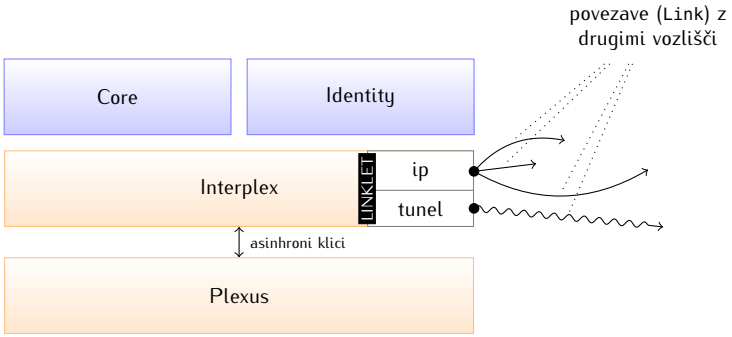
Generirane datoteke se nato uporabljajo skupaj z jezikovno-odvisno knjižnico (v primeru C++ verzije je to knjižnica `libprotobuf`). Za opis strukture sporočil se uporablja posebna sintaksa, ki je na hitro podobna jezikovnemu konstruktu `struct` iz C++. Podprti so številni osnovni podatkovni tipi, ki jih je mogoče tudi hierarhično sestavljati. Prevajalnik te datoteke prevede v ustrezno C++ kodo, ki omogoča izjemno hitro in učinkovito serializacijo ter deserializacijo sporočil.

4.3 Moduli in protokoli sistema

Na tem mestu se lotimo opisa zgradbe arhitekture ogrodja UNISPHERE. Ogrodje je zaradi preglednosti razdeljeno v več ločenih komponent, ki se nahajajo vsaka v svoji knjižnici. Komponente so naslednje:

- **Core:** Vsebuje osrednji razred za inicializacijo ogrodja, kontekst (`Context`), ki je vstopna točka ogrodja in povezuje vse ostale komponente.
- **Identity:** Gre za modul, ki je zadolžen za kriptografsko identifikacijo in avtentikacijo vozlišč.
- **Interplex:** Abstrakcija povezav med vozlišči vsebuje programski vmesnik za ustvarjanje povezav in izmenjavo ustrezno formatiranih sporočil. Vsa komunikacija je dogodkovno orientirana.
- **Plexus:** Modul, ki nadgradi *Interplex* z usmerjanjem preko prekrivnega omrežja, ki je izvedenka prej opisanega protokola Kademlia z nekaterimi spremembami.

Pogled na vse kompontente ogrodja s ptičje perspektive se nahaja na sliki 4.3. Vsaka izmed komponent živi v svoji knjižnici, katere ime je vedno oblike `libUnisphereXXX.so`, kjer je `XXX` naziv ustreznega modula (npr. `Core`). Ciljna aplikacija se mora nato samo povezati (angl. link) z omenjenimi knjižnicami ogrodja, kar pomeni da lahko aplikacija uporabi tudi samo delno funkcionalnost (za uporabo sta vedno odvisnosti modula `Core` in `Identity`, ki sta osnova za delovanje ogrodja). Tako je možno uporabiti tudi zgolj del za vzpostavljanje povezav in izmenjavo sporočil med vozlišči, brez da bi bilo potrebno uporabljati tudi prekrivno omrežje. Vsi razredi znotraj ogrodja se nahajajo v imenskem prostoru (angl. namespace) `UniSphere`.



Slika 4.3: Arhitektura ogrodja UNISPHERE. Modro so označeni moduli s pomožno funkcionalnostjo, oranžno pa moduli, ki implementirajo urejanje komunikacij in protokole za izmenjavo sporočil.

4.3.1 Core

Kot že omenjeno, modul *Core* povezuje vse komponente komunikacijskega ogrodja UNISPHERE. Samo ogrodje je zasnovano tako, da lahko znotraj aplikacije, ki ga uporablja, živi več različnih instanc konteksta. Kontekst tako vsebuje Boost.ASIO dogodkovne zanke (v obliki `asio::io_service` objekta), ki se uporabljajo za dispečiranje vseh sporočil med komponentami ogrodja, kontekst TLS, upravljalca s povezavami (ki bo podrobno opisan v modulu *Interplex*), instanco za zapis dnevnika in nenazadnje tudi identiteto vozlišča, ki je predstavljena z gradniki iz modula *Identity*.

Vse druge komponente torej za delo z vtičniki POSIX, za časovnike ter za ostale asinhrone klice uporabljajo dogodkovne zanke, ki jih daje na voljo jedrni modul preko konteksta. Ker je dostop do storitve iz konteksta zelo pogosta operacija, se reference na kontekst nahajajo v veliki večini razredov ostalih komponent.

4.3.2 Identity

Naslednji od modulov za osnovno funkcionalnost ogrodja se imenuje `Identity` in je namenjen implementaciji funkcij za delo s kriptografskimi identitetami vozlišč. Kot omenjeno že v poglavju o varnosti, je potrebno zagotoviti nekatere lastnosti, da zmanjšamo možnost napadov na omrežje in zato smo se odločili, da v omrežje vpeljemo certifikatno avtoriteto, ki vozliščem izdaja certifikate. Samo s temi certifikati lahko nato vozlišča vzpostavljajo povezave z drugimi vozlišči in si izmenjujejo sporočila.

Za strukturo certifikatov uporabljamo znani format X.509, ki definira nabor ter tipe posameznih polj v certifikatu in njihovo kodiranje. Za kodiranje certifikatov X.509 se ponavadi uporablja DER (angl. Distinguished Encoding Rules). Kodiranje DER je le eno izmed možnih kodiranj za specifikacijo struktur ASN.1 (angl. Abstract Syntax Notation One), kjer gre za podobno stvar kot je že prej omenjeni Googlov Protocol Buffers. Gre torej za način, kako specificiramo strukture sporočil in jih nato na različne načine zapišemo ter preberemo nazaj. Možni so tudi zapisi na druge načine (npr. XER omogoča zapis ASN.1 struktur v obliki XML), vendar je v kriptografiji pomembno, da so vse strukture vedno zapisane na enak način - brez tega sicer ne bi bilo mogoče preverjati digitalnih podpisov. Tako kodiranje DER zagotavlja, da se lahko dokument ASN.1 vedno zakodira na en sam način.

Za identifikatorje vozlišč v ogrodju UNISPHERE uporabljamo 160-bitne identifikatorje, ki so za predstavitev v certifikatih zaradi preglednosti predstavljeni v šestnajstiškem zapisu. Velikost 160 bitov pride iz uporabe kriptografske zgoščevalne funkcije SHA-1 za generiranje identifikatorjev. Če primerjamo to velikost z velikostjo naslovnega prostora protokolov IPv4 (32 bitov) in IPv6 (128 bitov), lahko ugotovimo, da nam takšna količina naslovov trenutno povsem zadošča. Teoretično zadostuje za 2^{160} vozlišč v kolikor bi identifikatorje uporabljali samo za vozlišča, ker pa jih, kot omenjeno v drugem poglavju, lahko uporabljamo za vrsto drugih stvari (od identifikatorjev skupin za oddajanje več prejemnikom do ključev za shranjevanje podatkov) pa je ta številka v praksi manjša. Hkrati nam uporaba omenjene zgoščevalne funkcije zagotavlja enakomerno razpršenost identifikatorjev po prostoru, kar je ključno za optimalnost usmerjanja.

Certifikatna avtoriteta

V našem ogrodju uporabljamo certifikate X.509 za predstavitev javnega ključa certifikatne avtoritete in za predstavitev javnih ključev ter identifikatorjev posameznih vozlišč. Modul `Identity` zato implementira osnovne funkcije za delo s certifikati in za preverjanje njihove veljavnosti. Razred `CertificateAuthority` predstavlja zbirko takšnih certifikatov in omogoča dodajanje zaupanih certifikatov CA preko knjižnice Botan ter nato preverjanje veljavnosti certifikatov vozlišč.

Poleg običajnega preverjanja poti zaupanja (angl. trust path), kjer gre za preverjanje podpisov od certifikata vozlišča do nekega certifikata, ki mu že zaupamo [9], tukaj preverjamo še ali se uporabljajo ustrezni kriptografski algoritmi (ogrodje zaenkrat zahteva uporabo najmanj 2048-bitnega RSA, vendar bo v prihodnosti to postalo bolj razširljivo tudi na druge algoritme). Preverja se tudi

časovna veljavnost certifikata (v primeru, da je certifikat vozlišča že potekel, le-ta ni več veljaven). Kot omenjeno že v razdelku 3.3.1, trenutno še ni implementiran mehanizem za porazdeljeno preverjanje preklicanih certifikatov.

Certifikati vozlišč

Vsako vozlišče potrebuje za svojo komunikacijo z ostalimi vozlišči veljaven certifikat, podpisan s strani certifikatne avtoritete. V ogrodju UNISPHERE za delo s certifikati vozlišč implementiramo razred `NodeCertificate`, ki vsebuje metode za nalaganje certifikatov iz različnih virov, metode za zapis certifikatov v kodiranje DER ter metodo za preverjanje veljavnosti certifikata v določenem kontekstu. Instance tega razreda se uporabljajo tako za predstavitev lokalnega certifikata vozlišča kot tudi za predstavitev certifikatov drugih vozlišč.

Omenili smo, da so certifikati vozlišč lahko pridobljeni iz različnih virov in zaradi tega potrebujemo tudi različne metode za njihovo dekodiranje:

- **Datoteka:** Metoda `fromFile` omogoča nalaganje certifikatov iz datotek v zapisu PEM. Pri tem zapisu gre v resnici za certifikat, kodiran v DER, vendar nato kodiran še s funkcijo Base64 [21], ki binarne vrednosti pretvori v znakovne ASCII vrednosti.

- **DER-kodirani podatki:** Metoda `fromData` omogoča nalaganje certifikatov neposredno iz nizov, ki predstavljajo certifikat v kodiranju DER. Ta metoda se uporablja za nalaganje certifikatov oddaljenih vozlišč s katerimi nimamo neposredne povezave.

- **TLS povezava z drugim vozliščem:** Metoda `fromSocket` pa je namenjena pridobivanju certifikatov iz že vzpostavljenih povezav preko protokola TLS. Ta način uporablja modul *Interplex* za preverjanje veljavnosti certifikatov vozlišč s katerimi je ravnokar vzpostavil sejo. V tej metodi gre v resnici za pretvorbo certifikata iz knjižnice OpenSSL (ki jo uporablja Boost.ASIO) v objekte Botan, ki jih uporablja tudi implementacija `NodeCertificate`. Pretvorba se vrši preko kodiranja certifikata v DER z vmesnikom OpenSSL in nato nalaganje tega certifikata v Botan.

Povedali smo, da veljavnost certifikata preverjamo v določenem kontekstu. Že pri opisu modula *Core* smo omenili, da je naše ogrodje zasnovano tako, da se lahko znotraj aplikacije nahaja več instanc konteksta, kar v resnici pomeni, da se lahko uporablja več neodvisnih komunikacijskih infrastruktur v eni sami aplikaciji. Vsak kontekst ima s sabo povezano tudi identiteto lokalnega vozlišča, ki je predstavljena kot trojka:

$$(nodeCertificate, nodePrivateKey, certificateAuthority)$$

Ta lokalna identiteta se hrani v instanci razreda `NodeIdentity`, ki mora biti prisotna preden se lahko ustvari kontekst ogrodja. Ko je kontekst ogrodja ustvarjen, je lokalna instanca identitete vedno na voljo preko konteksta z metodo `context.identity()`. Ker pa so certifikati vozlišč samostojni objekti in tako nepovezani s katerokoli eno instanco, se pri preverjanju veljavnosti kot parameter zahteva še kontekst preko katerega lahko pridemo do ustrezne certifikatne avtoritete. Popolnoma mogoče je namreč, da je nek certifikat v enem kontekstu veljaven, v drugem pa ne.

4.3.3 Interplex

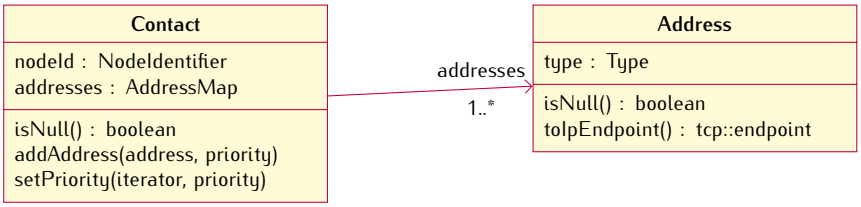
Z moduloma *Core* in *Identity* v resnici že imamo dovolj infrastrukture, da bi se lahko neposredno lotili implementacije prekrivnega omrežja z ovoji vtičnikov POSIX in sej TLS, ki so na voljo v knjižnici Boost.ASIO. Vendar smo se pri zasnovi arhitekture UNISPHERE odločili za drugačen pristop, ki olajša vzdrževanje, poveča robustnost in omogoča razširljivost.

Najprej smo se lotili razvoja modula imenovanega *Interplex*, ki predstavlja abstrakcijo komunikacije med vozlišči v obliki izmenjave ustrezno formatiranih sporočil, ki so predstavljena kot objekti razreda `Message`. Kot smo videli v razdelku 4.1.2, bo potrebno v ogrodju podpirati več različnih načinov povezav. Za modul prekrivnega omrežja bi bilo torej idealno, če je vmesnik za delo s povezavami enak ne glede na to kakšna vrsta povezave se dejansko uporablja za komunikacijo.

Prav tako je lahko vzpostavljanje povezav kompleksno opravilo (v smislu števila operacij in prehodov med stanji), ki zahteva prehod med različnimi stanji in reakcijo na veliko različnih dogodkov. Zaradi tega se nam je zdel boljši pristop, da ta del komunikacije ločimo od samega prekrivnega omrežja. Hkrati to omogoča, da se modul *Interplex* lahko uporablja tudi povsem ločeno od prekrivnega omrežja za vzpostavljanje komunikacijskih kanalov med vozlišči, hkrati pa razvijalec zraven zastonj dobi obstoječo avtentikacijsko infrastrukturo.

Kontaktni podatki vozlišč

Preden pa se lotimo samih povezav potrebujemo predstavitev za vse različne kontaktne podatke. Vsako vozlišče je namreč lahko dosegljivo na več različnih naslovih, ki so lahko različnih tipov. Vse kontaktne naslove vozlišča predstavljamo z razredom `Contact` (glej sliko 4.4), ki vsebuje identifikator vozlišča in



Slika 4.4: Razredni diagram struktur za hranjenje kontaktnih podatkov.

slovar vseh znanih naslovov. Uporabljeni slovar je vrste `std::multimap`, kar pomeni da lahko en sam ključ vsebuje več vrednosti, uporablja pa se za možnost določitev prioritete posameznim naslovom. Ta prioriteta se potem upošteva pri vzpostavljanju povezav in jo je mogoče spremeniti glede na izkušnje pri zanesljivosti kontaktiranja na določenem naslovu.

Vsak posamezni naslov se nahaja v objektih razreda `Address`, ki omogoča hranjenje različnih tipov naslovov. Enumeracija `Address::Type` določa vrsto naslova in sicer so trenutno podprti neposredni naslovi IP ter navidezni naslovi v obliki poti skozi vozlišča, ki se uporabljajo za izvorno usmerjanje.

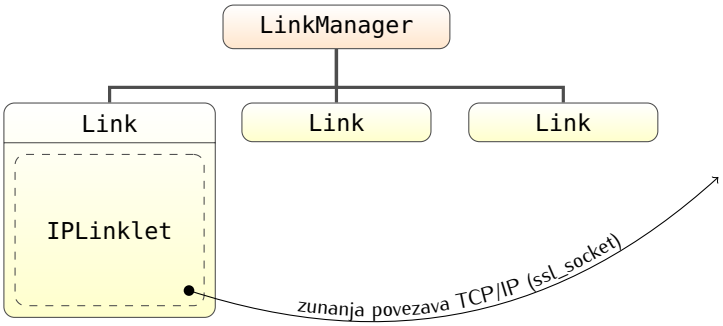
Povezave in povezovalniki

Ključna lastnost modula *Interplex* pa je abstrakcija dela s povezavami. Vodilo pri razvoju je bilo, da mora biti za razvijalce (in tudi kasneje za modul *Plexus*) zunanji vmesnik zastavljen čim bolj preprosto, vsi detajli vzpostavitve in vzdrževanja povezav ter prenosa sporočil pa morajo biti skriti. Razvijalec naj bi tako moral za pošiljanje sporočil napisati zgolj nekaj vrstic:

```
1 Contact contact(NodeIdentifier("83d421...db49334426"));
2 contact.addAddress(Address("127.0.0.1", 8472));
3 Link &link = Link::connect(context, contact);
4 link.send(msg);
```

Slika 4.5: Pošiljanje sporočila drugemu vozlišču z uporabo modula *Interplex*.

V kodi s slike 4.5 se skriva še en detajl in sicer v vrstici 4 pošljemo sporočilo še preden je povezava sploh vzpostavljena (namreč metoda `connect` zgolj pripravi vse potrebno, ker pa ogrodje deluje asinhrono pa se vzpostavitev zgodi šele kasneje). Naš vmesnik nam omogoča takojšnje pošiljanje sporočil, v primeru da povezava ni vzpostavljena pa bodo sporočila zadržana v vrsti in poslana takoj, ko bo to mogoče.



Slika 4.6: Relacije med razredi za nadzor nad povezavami v modulu *Interplex*.

Sedaj se lahko lotimo opisa arhitekture, ki stoji za zgornjo kodo, ki je precej skopa z detajli dogajanja v ozadju. V modulu za neposredno komunikacijo med vozlišči smo implementirali tri skupine pomembnih razredov, ki skupaj omogočajo zgornjo funkcionalnost. Ti razredi so naslednji:

1. **Link:** Znan razred s slike 4.5 z instancami katerega so predstavljeni objekti abstraktnih povezav, ki so vidni tudi uporabniku ogrodja. Ponuja vmesnik za vzpostavitev in prekinitev povezave, ter za pošiljanje in sprejemanje sporočil. Ob dogodkih proži različne signale na katere se lahko uporabnik ogrodja naroči (te signale uporablja tudi modul *Plexus*).

2. **Linklet:** Povezovalnik je abstraktni razred, ki ga morajo implementirati vse konkretne implementacije povezav. Gre torej za razred, ki dela neposredno z vtičniki POSIX oz. s sejami TLS. Obstajajta dve vrsti povezovalnikov, eni implementirajo seje TLS (`IPLinklet`), drugi pa tunele preko drugih vozlišč (`TunnelLinklet`).

3. **LinkManager:** Skrbi za hranjenje referenc do vseh povezav, ki so trenutno aktivne znotraj konteksta. Je lastnik (angl. owner) kazalcev vseh objektov `Link`, ki za nove povezave poslušajo. Za povezave je mogoče poslušati s preprostim klicem statične metode `Link::listen(context, Address("127.0.0.1", 8473));`.

Povezave v ogrodju UNISPHERE so torej dvodelne, na eni strani imamo uporabniški del v obliki razreda `Link`, na drugi strani pa konkretno implementacijo komunikacije v obliki podrazredov `Linklet`. Celoten odnos med razredi je predstavljen v sliki 4.6. Razreda `Linklet` in `Link` med seboj komunicirata s pomočjo signalov, ki se prožijo ob določenih dogodkih. Vsaka povezava ima v resnici lahko večje število povezovalnikov, v kolikor so vsi povezovalniki povezani z istim vozliščem.

Dostava sporočil

Ko imamo povezave vzpostavljene, lahko začneta vozlišči med sabo komunicirati z izmenjevanjem sporočil. Kot smo že omenili, morajo biti vsa sporočila v ogrodju UNISPHERE veljavna Protocol Buffers sporočila (ustrezni objekti razreda `google::protobuf::Message`). Naše ogrodje ponuja preprost ovoj okrog Protocol Buffers struktur, ki doda še enostavno glavo, v kateri se nahaja velikost sporočila in njegov tip (angl. typed size-prefixed blob). Celotna glava je velika 40 bitov, kjer je prvih 32 bitov namenjenih specifikaciji dolžine, zadnjih 8 pa tipu sporočila. Vse številske vrednosti so predstavljene po pravilu debelega konca (angl. big endian). Glavi sledi ustrezna količina podatkov, serializiranih v formatu Protocol Buffers.

Razred ogrodja, ki predstavlja sporočilo (`Message`) implementira serializacijo in deserializacijo sporočil v dveh stopnjah. Najprej se odpakira glava sporočila, nato pa se rezervira ustrezni del pomnilnika za sprejetje sporočila. Nato se knjižnici Boost.ASIO neposredno poda naslov dela pomnilnika s sporočilom, kamor se kasneje prenese serializirano sporočilo. Deserializacija je seveda odvisna od tipa sporočila, saj zanjo potrebujemo ustrezno generiran deserializator, ki ga je generiral prevajalnik `protoc`. Za lažje delo s sporočili naše ogrodje implementira funkcijo `message_cast<T>`, ki sprejme instanco `Message` ter ciljni deserializator in vrne ustrezno novo instanco deserializiranega sporočila.

Dekodiranje sporočil se vrši avtomatsko ob prejetju v ustreznem `Linklet` razredu, vsa prejeta sporočila pa se nato posredujejo preko razreda `Link` in njegovega signala `signalMessageReceived`. Pri pošiljanju sporočil so stvari malo bolj zapletene, saj je povsem možno, da za eno povezavo obstaja večje število povezovalnikov, vprašanje pa je čez kateri povezovalnik poslati sporočilo. Druga težava pa je, da se lahko zgodi, da na voljo ni nobenega povezovalnika, ki bi že uspešno vzpostavil povezavo. Za ta drugi primer imajo vse povezave vgrajeno kratko vrsto sporočil, v katero jih lahko hranijo v kolikor trenutno še ne morejo biti poslana (v primeru, da se vrsta zapolni so sporočila zavržena). Za prvi primer pa ogrodje abstrahira strategijo pošiljanja sporočil skozi več povezovalnikov. Trenutno je implementiran zgolj `RoundRobinMessageDispatcher`, ki posreduje sporočila izmenično skozi vse povezovalnike, ki imajo vzpostavljene povezave.

Vzpostavitev povezave

Vzpostavitev povezave v modulu *Interplex* gre skozi različne faze, ki se malo razlikujejo glede na to ali gre za vzpostavljanje povezave z oddaljenim vozliščem v odjemalskem načinu ali pa se drugo vozlišče povezuje na naše vozlišče in je

```
1 message Hello {
2   required UniSphere.Protocol.Contact local_contact = 1;
3 }
```

Slika 4.7: Zgradba pozdravnega sporočila v protokolu za vzpostavitev povezav *Interplex*.

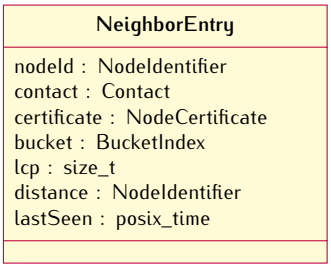
povezava v strežniškem načinu. Za razvijalca, ki uporablja naše ogrodje je v resnici popolnoma vseeno v katero smer se povezava vzpostavi, saj so zanj vsi ti detajli nevidni. Razvijalec vidi le vzpostavljeno povezavo med dvema vozliščema po kateri lahko pošilja sporočila.

V primeru, da je povezovalnik v *odjemalskem načinu*, povezava začne v stanju `Closed` in se takoj po pridobitvi kontaktnih podatkov prestavi v stanje `Connecting`. Tukaj se izvede standardno trojno rokovanje protokola TCP/IP, nato pa ogrodje izvede še začetek seje TLS, ki zajema tudi preverjanje certifikatov. Ko so certifikati preverjeni in so veljavni, preide povezava v stanje `IntroWait`, v katerem vsako izmed vozlišč posreduje poseben paket tipa `Interplex_Hello`, ki vsebuje lokalne kontaktne podatke (glej sliko 4.7). Dokler si vozlišči ne izmenjata svojih kontaktnih podatkov, čez povezavo ni mogoče pošiljati sporočil, ki niso *pozdravna sporočila*. Te kontaktne podatke nato vozlišči dodata v svoje deskriptorje povezav za bolj zanesljivo komunikacijo v prihodnosti, na voljo pa so tudi višjenivojskim modulom (npr. modul *Plexus* uporablja te podatke za dodajanje vnosov v usmerjevalno tabelo). Po uspešni izmenjavi povezavi preideta v stanje `Connected`.

Pri *strežniškem načinu* je stvar zelo podobna, edina razlika je ta, da se na začetku uporabi povezovalnik v stanju `Listen`, katerega vloga je zgolj to, da sprejema prihajajoče (angl. inbound) povezave in za vsako izmed njih ustvari nov povezovalnik tipa `IPLinklet`. Ko je tak povezovalnik ustvarjen, prej omenjeni `LinkManager` poskrbi za vstavljanje tega povezovalnika v ustrezno povezavo. Ta povezava je lahko že odprta (v kolikor se je vozlišče A že povezalo na vozlišče B in se nato še vozlišče B poveže na vozlišče A) in v tem primeru bosta oba povezovalnika združena v eni povezavi (pri dostavi sporočil se bo, kot že omenjeno, uporabljal *round-robin* princip).

4.3.4 Plexus

Zadnji modul, imenovan *Plexus*, je namenjen povezavi vseh prej obstoječih modulov in njihovi nadgradnji s prekrivnim omrežjem, ki omogoča usmerjanje spo-



Slika 4.8: Struktura, ki predstavlja vnos enega vozlišča v usmerjevalni tabeli.

ročil med poljubnimi pari vozlišč. Kot protokol prekrivnega omrežja je bil izbran protokol Kademlia, osrednjo strukturo katerega predstavlja usmerjevalna tabela zgrajena iz k-veder. V razdelku 2.2.4 na sliki 2.8 smo videli, da gre v resnici za hierarhično strukturo, ki ima v listih sezname kontaktnih podatkov vozlišč.

Usmerjevalna tabela je v ogrodju UNISPHERE implementirana v razredu `RoutingTable`, vendar za svoje delovanje ne uporablja dejanske drevesne strukture. Drevesna struktura v obliki drevesa predpon bi bila mogoče najbolj očitna predstavitev, vendar se pri implementaciji hitro pokaže, da je potrebno vozlišča v usmerjevalni tabeli sortirati po različnih kriterijih. V ta namen pride zelo prav podatkovna struktura, ki je del knjižnice Boost in se imenuje mnogo-indeksni kontejner (angl. multi-index container). Koncept podatkovne strukture je zelo podoben tistemu, ki ga poznamo iz sistemov za upravljanje z relacijskimi podatkovnimi bazami, kjer imamo tabelo z vrsticami nad katero lahko namestimo različne indekse za hitrejše poizvedbe po določenih ključih.

Za podoben princip gre tudi v primeru naše implementacije usmerjevalne tabele, vsi vnosi v njej so namreč predstavljeni v ploščati strukturi, ki je enakovredna seznamu z naključnim dostopom (npr. `std::vector`). Vsak vnos je instanca razreda `NeighborEntry`, ki vsebuje vsa potrebna polja za povezavo z vozlišči in vzdrževanje usmerjevalne tabele (glej sliko 4.8). Poleg shranjevanja samih podatkov v strukturi, pa ima le-ta na voljo tudi indekse po določenih poljih. Sama definicija indeksov je v jeziku C++ izvedena s pomočjo šablon (angl. templates), del definicije tipa `NeighborTable` izgleda takole:

```
1 typedef boost::multi_index_container<
2   NeighborEntry,
3   midx::indexed_by<
4     // Index by node identifier
5     midx::ordered_unique<
6       midx::tag<NodeIdTag>,
7       BOOST_MULTI_INDEX_MEMBER(NeighborEntry, NodeIdentifier, nodeId)
```

```
    >,

    // Index by entry's k-bucket
    midx::ordered_non_unique<
      midx::tag<BucketIndexTag>,
      BOOST_MULTI_INDEX_MEMBER(NeighborEntry, BucketIndex, bucket)
    >,

    // Index by entry's k-bucket and sorted by distance within the bucket
    midx::ordered_non_unique<
      midx::tag<BucketByDistanceTag>,
      midx::composite_key<
        NeighborEntry,
        BOOST_MULTI_INDEX_MEMBER(NeighborEntry, BucketIndex, bucket),
        BOOST_MULTI_INDEX_MEMBER(NeighborEntry, NodeIdentifier, distance)
      >
    >,

    ...
```

Ta podatkovna struktura nam potem zagotavlja samodejno gradnjo in posodobitev indeksov ob spremembah polj posameznih vnosov, ki so del kakšnega izmed indeksov in tako vplivajo na njegovo strukturo. Naša usmerjevalna tabela nad to strukturo implementira nekaj operacij, ki omogočajo uporabo le-te pri usmerjanju paketov. Vmesnik, ki je viden navzven vsebuje tri osnovne operacije in sicer:

- `add(contact, certificate)` je operacija namenjena vstavljanju novih kontaktnih podatkov vozlišč v usmerjevalno tabelo. Ta operacija je v resnici sestavljena iz različnih podoperacij, ki so odvisne od stanja v posameznih k-vedrih.
- `lookup(destination, count)` omogoča poizvedbo za *count* najbližjih sosedov danemu identifikatorju v prostoru ključev, kjer se za mero razdalje uporablja metrika XOR.
- `get(nodeId)` pa je namenjen direktni poizvedbi po kontaktnih podatkih vozlišča s točno določenim identifikatorjem, v primeru da podatkov za to vozlišče v usmerjevalni tabeli ni, pa vrne prazno kontaktno strukturo.

Posebno pozornost moramo nameniti operacijama `add` in `lookup`, saj implementirata bistvo delovanja prekrivnega omrežja Kademlia. Za k-vedra smo v

našem ogrodju torej uporabili indeksno naslavljanje (vsakemu k-vedru je dodeljna neka zaporedna številka, ki predstavlja število skupnih bitov z lokalnim vozliščem v predponi identifikatorja). Na začetku obstaja v usmerjevalni tabeli zgolj k-vedro pod zaporedno številko 0. Tja padejo vsa vozlišča, ki imajo z identifikatorjem lokalnega vozlišča vsaj 0 skupnih bitov (kar pomeni, da dejansko na začetku tja pade prav vsako vozlišče, ki je dodano z operacijo `add`).

Ko se vedro napolni do dolžine k, ga je potrebno razdeliti v dve sosednji vedri. Razdeljevanje veder v naši strukturi izgleda zelo preprosto - če je do sedaj lokalno vozlišče padlo v vedro 0, bo po razdeljevanju padlo v vedro 1 (razdeljuje se namreč vedno samo k-vedro za katerega pravimo, da pokriva lokalno vozlišče, kar je vedno vedro, ki ima z lokalnim vozliščem največ skupnih bitov in ima torej najvišji indeks). Pri razdeljevanju vedra b_i moramo preiskati vsa obstoječa vozlišča v vedru b_i in v novo vedro b_{i+1} prenesti vsa tista vozlišča za katera velja $lcp(localId, nodeId) >= i + 1$, kjer funkcija $lcp(x, y)$ vrne število bitov z leve, ki so skupni identifikatorjema x in y. Funkcija za primerjavo predpon je v našem ogrodju implementirana v razredu `NodeIdentifier` v obliki metode `longestCommonPrefix`, ki po bajtih izračuna vrednosti funkcije XOR nato pa poišče indeks prvega bita, ki je enak 1. To je namreč tudi prvi bit v katerem se identifikatorja začneta razlikovati.

Poleg razdeljevanja ciljnega k-vedra sta mogoča še dva scenarija, vedro je lahko polno, ne vsebuje lokalnega vozlišča in novo vstavljeno vozlišče ni del k-okolice lokalnega vozlišča. V tem primeru moramo na novo dodano vozlišče v k-vedru zavreči. Bolj zanimiv primer pa se pojavi v kolikor je na novo vstavljeno vozlišče v resnici del k-okolice, ne pade pa v vedro, ki trenutno pokriva lokalno vozlišče. Zaradi tega ciljnega vedra ni mogoče razdeliti, ker pa je pomembno, da vsako vozlišče pozna svojo k-okolico (to je pomembno zaradi replikacije ključev in vzdrževanja konsistence usmerjevalne tabele), pa takega vozlišča ne moremo preprosto zavreči. V tem primeru moramo vozlišče vseeno vstaviti v ciljno k-vedro in tako ohraniti znanje o k-okolici. Seveda moramo najprej ugotoviti, da vozlišče res pade v k-okolico lokalnega vozlišča - to naredimo z uporabo operacije `lookup`, ki nam vrne najbližjih k vozlišč v bližini lokalnega identifikatorja.

Druga pomembna operacija v usmerjevalni tabeli je torej `lookup`, ki omogoča iskanje vozlišč v neposredni okolici (po razdalji XOR) določenega identifikatorja. Implementacija je preprosta, vozlišča začnemo zbirati v k-vedru v katerega pade identifikator, nato pa se sprehajamo po vedrih (najprej desno in nato levo) dokler nimamo zajetiv vsaj n potencialnih vozlišč. Nato ta vozlišča sortiramo po razdalji XOR in najbližjih n vrnemo kot rezultat operacije.

Vse operacije pri izvajanju uporabljajo deljeni mutex (angl. shared mutex),

ki omogoča dve vrsti dostopa do kritičnega območja (deljeni in izključni). Operacije, ki iz strukture samo berejo pridobijo deljeni dostop, kar pomeni, da lahko vzporedno poteka več operacij, ki samo berejo. Kratka območja, kje se dogajajo spremembe usmerjevalne tabele, pa pridobijo izključni dostop, kar pomeni da takrat ni dovoljeno ne branje ne pisanje v strukturo. Vse to je potrebno za zagotovitev uporabe usmerjevalne tabele v večnitnih okoljih.

Vmesnik za klice RPC

Za izmenjavo usmerjevalnih informacij med vozlišči je potrebno med njimi zagotoviti mehanizem za izvajanje klicev RPC (angl. remote procedure calls). Gre za koncept odziva na zahteve, kjer kličemo metode na oddaljenem vozlišču kot bi klicali metode lokalnega razreda. V ta namen modul *Plexus* implementira preprost vmesnik, ki je predstavljen v razredu `RpcEngine`. Preko tega razreda je mogoča tako registracija metod, ki jih vozlišče daje na voljo kot tudi izvajanje klicev. Vsa sporočila so seveda predstavljena v formatu Protocol Buffers.

```
1  package UniSphere.Protocol;
2
3  //
4  // RPC request message
5  //
6  message RpcRequest {
7    required fixed64 rpc_id = 1;
8    required string method = 2;
9    required bytes data = 3;
10 }
11
12 //
13 // RPC response message
14 //
15 message RpcResponse {
16   required fixed64 rpc_id = 1;
17   required bytes data = 2;
18 }
```

Slika 4.9: Sporočili za protokol RPC med vozlišči.

Vsaka metoda RPC je identificirana z nizom, ki predstavlja njeno ime (polje `method`). To ime se uporablja pri vseh klicih te specifične metode. Drugo

pomembno polje je `rpc_id`, kjer gre za naključni 64-bitni identifikator posameznega zahtevka s katerim lahko povežemo odgovor, ki sledi specifičnemu klicu metode. Za vodenje evidence klicev in njihovih povezav z unikatnimi identifikatorji prav tako skrbi razred `RpcEngine`. Vsebina zahteve ali odgovora se nahaja v polju `data`, ki vsebuje ustrezno serializirano Protocol Buffers sporočilo, katerega struktura je odvisna od metode, ki jo kličemo. Registracija in klic sta v ogrodju implementirana s pomočjo šablon in funkcij lambda, tako da je sintaksa za razvijalca čim bolj enostavna.

```
1 using namespace Protocol::Routing;
2
3 // Register the Core.Routing.NextHops RPC method
4 m_rpc->registerMethod<NextHopsRequest, NextHopsResponse>(
5   "Core.Routing.NextHops", [](LinkPtr link, const NextHopsRequest &request)
6     -> NextHopsResponse {
7     // ...
8   }
9 );
```

Slika 4.10: Način registracije metode RPC s pomočjo uporabe `RpcEngine`.

Primer uporabe ogrodja za klice RPC nam daje koda za registracijo nove metode, ki bo po registraciji dostopna preko mehanizma RPC in je del kode za postopke usmerjanja (slika 4.10). Ob prejetju sporočil razred `RpcEngine` skrbi za izvedbo ustreznih metod in posredovanju odgovorov nazaj skozi povezavo preko katere so prišli. Vsa serializacija oz. deserializacija klicev metod v ustrezna sporočila, ki jih lahko okrog posreduje infrastruktura modula *Interplex*, se dogaja v ozadju transparentno za razvijalca.

Usmerjanje sporočil in grajenje poti

Naslednje pomembno opravilo, ki je tudi cilj modula *Plexus*, je dejansko usmerjanje sporočil preko prekrivnega omrežja. Da bi bilo usmerjanje mogoče, pa je najprej potrebno med vozlišči zgraditi ustrezne povezave in tako vzpostaviti poti. Dostava sporočil se vedno vrši neposredno preko povezave (objekta tipa `Link`), kar pomeni da mora za uspešno dostavo lokalno vozlišče s ciljnim vzpostaviti povezavo. Na začetku lokalno vozlišče ponavadi nima neposredno kontaktnih podatkov ciljnega vozlišča, zato jih mora pred uspešno komunikacijo najprej pridobiti preko prekrivnega omrežja.

```
message NextHopsRequest {
  required bytes destination = 1;
  required uint32 count_hops = 2;
  optional bool include_target = 3 [default = true];
}

message NextHopsResponse {
  repeated UniSphere.Protocol.Contact contacts = 1;
}
```

Slika 4.11: Sporočili za klic RPC metode `Core.Routing.NextHops`, ki jo uporabljajo postopki grajenja poti.

Osrednji navzven viden vmesnik, ki ga ponuja naša implementacija razreda `Router`, je predstavljen skozi metodo `sendMessage`. Ta metoda preprosto sprejme ciljni identifikator in sporočilo, ki naj bo poslano. Metoda najprej ustvari novo instanco `Link` za ciljno vozlišče in skoznjo pošlje sporočilo. Seveda na začetku take povezave sploh še ni in v resnici je takšna povezava brez kontaktnih podatkov in torej brez povezovalnikov. Kot že omenjeno pri opisu modula *Interplex*, se sporočila v takem primeru postavijo v vrsto. V kolikor kontaktnih podatkov za ciljno povezavo še ni, se pokliče metoda `buildPath`, ki je zadolžena za grajenje dejanske poti do cilja. Grajenje poti je popolnoma asinhron postopek, ki ima za rezultat (v kolikor je grajenje uspešno) kontaktne podatke za ciljno povezavo. Ti kontaktni podatki se nato preprosto dodajo obstoječi povezavi, infrastruktura *Interplex* pa poskrbi za vzpostavitev ustreznih povezovalnikov, izvedbo povezave in pošiljanje sporočil, ki čakajo v vrsti. Ključni del usmerjanja skozi prekrivno omrežje se torej skriva v metodi za gradnjo poti.

Pridobivanje kontaktnih podatkov poteka po korakih in sicer glede na približevanje ciljnemu identifikatorju s čim daljšimi skoki preko prostora identifikatorjev, ki je določen z metriko XOR. Grajenje poti začnemo z vozliščem, ki je ciljnemu identifikatorju najbližje po omenjeni razdalji v lokalni usmerjevalni tabeli. Nato približevanje izvajamo v korakih, kjer v naši implementaciji za en tak korak skrbi metoda `buildPathHop`, ki je zadolžena za vzpostavitev povezave naslednjega koraka. Metoda najprej preveri ali smo že prišli do cilja in v kolikor še nismo, izvede nad naslednjim vozliščem v verigi (to je vozlišče, ki smo ga ravnokar pridobili iz enega izmed k-veder usmerjevalne tabele) klic RPC metode `Core.Routing.NextHops`. Na sliki 4.11 so vidni parametri te metode in njen rezultat - kot parameter prejme ciljni identifikator in največje število najbližjih kontaktov, ki naj jih metoda vrne.

V kolikor je poizvedovanje uspešno, se tak klic RPC vrne z ustreznim številom kontaktnih podatkov iz svojega k-vedra (trenutno se za ta parameter vedno uporablja en sam kontakt, vendar obstaja možnost nadgraditve s paralelizacijo poizvedb). Metoda za grajenje trenutnega koraka poti nato zahteva grajenje povezave do na novo pridobljenega kontakta, in v kolikor je le-ta uspešna, postane naslednji člen v verigi na novo spoznano vozlišče. Nad tem novim vozliščem se nato spet pokliče metoda `buildPathHop`. Ta postopek se nadaljuje vse dokler ne ugotovimo da ali ciljnega identifikatorja ni v usmerjevalni tabeli ali pa vzpostavimo povezavo do njega. V tem primeru se samodejno dostavijo tudi vsa sporočila v vrsti.

Implementacija metode `buildPathHop` je izvedena s pomočjo lambda funkcij, ki so konstrukt jezika C++11 in v našem primeru omogočajo pregledno izvedbo asinhronih operacij (vse zgoraj omenjene operacije se namreč izvajajo glede na zunanje dogodke, kot že omenjeno pri opisu delovanja arhitekture Boost.ASIO). Ključni izsek implementacije te metode se nahaja spodaj:

```
1  // Request next hop from our waypoint
2  NextHopsRequest request;
3  request.set_destination(destination.as(NodeIdentifier::Format::Raw));
4  request.set_count_hops(1);
5
6  m_rpc->call<NextHopsRequest, NextHopsResponse>(hop, RoutingType::Direct,
7    "Core.Routing.NextHops", request, [=](const NextHopsResponse &response) {
8      // RPC was a complete success, build link to destination
9      if (!response.contacts_size()) {
10       UNISPHERE_LOG(m_context, Warning, "Router: Path building has failed "
11         "due to no route to destination!");
12       return;
13     }
14
15     Contact nextHop = Contact::fromMessage(response.contacts(0));
16     buildLink(nextHop.nodeId(), hop, [=]() {
17       // Link has been successfully built, proceed with next hop
18       buildPathHop(destination, nextHop.nodeId());
19     }, [=]() {
20       // Failure to establish a link
21       UNISPHERE_LOG(m_context, Warning, "Router: Path building has failed "
22         "due to down link!");
23     });
24   }, [=]() {
25     UNISPHERE_LOG(m_context, Warning, "Router: Path building has failed "
```

```
26        "due to RPC failure!");
27    }, 10
28 );
```

Pridružitev omrežju in združitev usmerjevalnih tabel

Zadnja pomembna operacija za modul *Plexus* je vzpostavitev povezave s preostalim omrežjem, ko imamo prazne usmerjevalne tabele (angl. bootstrapping). Hkrati smo v razdelku 4.1.2 že omenili, da je potrebno narediti to pridruževanje bolj robustno. V protokolu našega prekrivnega omrežja (ki je povzet po [14]) zato implementiramo operacijo `merge`, ki se izvede vsakokrat, ko se vzpostavi kakšna nova povezava med dvema vozliščema. Ta operacija asinhrono posodobi k-vedra na tak način, da je še naprej mogoče neprekinjeno usmerjanje preko omrežja v kolikor poti obstajajo. V kolikor se vozlišče A pridruži omrežju vozlišča B, A najprej zapolni spodnjih $d = lcp(id_A, id_B)$ k-veder (ker imata vozlišči skupno predpono dolžine d bitov, so vnosi v teh k-vedrih primerni za obe vozlišči) in vzpostavi ustrezne povezave. Konaktne podatke pridobi preko klica metode `Core.Routing.SampleBuckets`, ki se izvede preko mehanizma RPC.

Nato vozlišče A zapolni še vedro v katerega pade vozlišče A v usmerjevalni tabeli vozlišča B v kolikor ima vozlišče B v njem še kakšen vnos, ki je različen od vozlišča B. To naredi podobno kot pri grajenju poti, s klicem metode RPC `Core.Routing.NextHops`, le da tokrat uporabi za ciljni identifikator svoj lastni identifikator in nastavi parameter `include_target` na `false` (slika 4.11), saj ne želimo pridobiti svojih lastnih kontaktnih podatkov. V primeru, da tak vnos obstaja (recimo mu vozlišče C), se celoten postopek ponovi še za vozlišče C, vse dokler na koncu ne pridemo do vozlišča, ki ima v vedru kamor pade vozlišče A, samo vnos za vozlišče A.

Seveda operacijo `merge` izvede tudi vozlišče B nad vozliščem A, in tako sta vozlišči A in B sedaj dostopni iz obeh potencialnih particij omrežja. Podobno operacijo morajo izvesti tudi ustrezna sosednja vozlišča, zato v primeru spremembe vnosov v usmerjevalni tabeli tekom postopka `merge` obe vozlišči obvestita svoje sosede, ki potem tudi sama izvedejo metodo `merge`. Obvestila sosedov so v resnici posebna sporočila tipa `Plexus_MergeNotify`.

Poglavje 5

Testiranje

Za preizkus naše implementacije smo pripravili testno aplikacijo, ki za komunikacijo uporablja komponente našega ogrodja. Ker ogrodje UNISPHERE za svoje delovanje potrebuje vzpostavljeno certifikatno avtoriteto in ustrezno izdane certifikate, smo najprej pripravili še enostavno orodje za generiranje in izdajanje ustreznih certifikatov ter zasebnih ključev. Orodje je izvedljivo iz ukazne vrstice in omogoča dve operaciji:

- **Generiranje nove certifikatne avtoritete:** Klic s stikalom `--create-ca` generira nov ključ RSA in samo-podpisan (angl. self-signed) certifikat v formatu X.509 za certifikatno avtoriteto. Rezultat sta torej dve datoteki, ena vsebuje zasebni ključ, druga pa javni ključ in certifikat avtoritete.

- **Izdajanje certifikata vozlišča:** Z uporabo stikala `--create-node` pa je mogoče generirati tudi ustrezne certifikate in zasebne ključe za posamezno vozlišče. Pred tem je seveda treba imeti dostop do veljavnega zasebnega ključa certifikatne avtoritete. Orodje zgenerira zasebni ključ in zahtevek za izdajo certifikata, nato pa ga s pomočjo zasebnega ključa certifikatne avtoritete tudi generira in podpiše. Tak način izdajanja certifikatov vozlišč sicer ni pretirano dober z varnostnega stališča, vendar je za potrebe testiranja vseeno ustrezen.

S pomočjo tega orodja smo nato zgenerirali certifikatno avtoriteto in s pomočjo skripte (slika 5.1) tudi 1000 različnih certifikatov vozlišč, ki smo jih kasneje uporabili za testiranje. Naša testna aplikacija (imenovana `unisphered`) prav tako sprejme nekaj parametrov preko ukazne vrstice, ki določajo način delovanja same aplikacije. Da se lahko posamezno novo vozlišče vključi v omrežje, mora namreč poznati identifikator in kontaktne podatke vsaj še enega drugega

```
#!/bin/bash
./unisphere_cert --create-ca --password foo --ca-certificate unisphere.ca \
  --ca-private unisphere.key

for i in $(seq 1 1000); do
  ./unisphere_cert --create-node --password foo \
    --ca-certificate unisphere.ca --ca-private unisphere.key \
    --node-certificate node$i.crt --node-private node$i.key
done
```

Slika 5.1: Skripta za generiranje testnih certifikatov vozlišč.

vozlišča. Ko so taki podatki znani, lahko preprosto uporabi metodo razreda `Router` imenovano `link`, katera sprejme instanco `Contact` in vzpostavi povezavo s ciljnim vozliščem. Ker se ob vzpostavitvi povezave samodejno izvedejo funkcije za združevanje in uskladitev usmerjevalnih tabel, je novo vozlišče nato zelo hitro pridruženo v omrežje, brez da bi bilo potrebno razvijalcu storiti še kaj.

Celotni postopek vzpostavitve vozlišča je zato za uporabinka ogrodja zelo preprost:

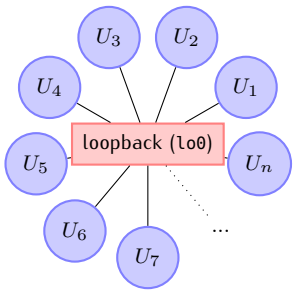
```
NodeCertificate cert = NodeCertificate::fromFile("node.crt");
NodePrivateKey key = NodePrivateKey::fromFile("node.key");

// Setup the certificate authority
CertificateAuthority ca;
ca.addCertificate("unisphere.ca");

// Bind everything into a node identity
NodeIdentity identity(cert, key, ca);
Context context(identity);
Router router(context);

// Start listening and establish a bootstrap link when requested
Link::listen(context, Address("127.0.0.23", 8472));
Contact bootstrap(NodeIdentifier("7eac37eac37eac3...7eac3",
  NodeIdentifier::Format::Hex));
bootstrap.addAddress(Address("127.0.0.1", 8472));
router.link(bootstrap);

context.run();
```

Ko pa je vozlišče enkrat pridruženo omrežju lahko uporablja klice metode `router.sendMessage` za dostavo sporočil oz. v kolikor želi uporabljati vmesnik RPC, lahko uporabi metodo `router.rpcEngine().call`, ki skrbi za ustrezno formatiranje in posredovanje sporočil.

5.1 Testno okolje in topologija

Pri testiranju smo uporabili lokalni vmesnik (angl. loopback) omrežnega sklada Linux za različne naslove vozlišč (območje od `127.0.0.1` do `127.0.3.238`). Lokalni vmesnik je navidezna omrežna naprava z oznako `lo0`, ki omogoča izmenjavo sporočil med aplikacijami znotraj iste instance operacijskega sistema. V našem primeru smo jo uporabili zato, da smo lahko simulirali veliko število priklopljenih vozlišč, vsako s svojim naslovom IP.

Vozlišče `127.0.0.1` (oz. če gledamo datoteke s certifikati je bilo to vozlišče imenovano `node1`) smo proglasili za ti. *bootstrap* vozlišče. To je vozlišče na katerega so se prvič povezana druga vozlišča za začetno uskladitev svojih usmerjevalnih tabel (na začetku ima namreč vsako vozlišče povsem prazno usmerjevalno tabelo in da bi se uspešno pridružilo omrežju mora le-to poznati vsaj naslov enega drugega vozlišča, ki je že del omrežja). Takoj po uskladitvi usmerjevalnih tabel so se vozlišča povezala s svojimi sosedi in tako vzpostavila svoje neodvisne povezave.



Slika 5.2: Prikaz preproste testne topologije omrežja v obliki zvezde povezane preko vmesnika `lo0`.

Topologija omrežja IP, ki se je uporabljalo za potrebe testiranja (glej sliko 5.2) je bila v našem primeru relativno preprosta. Vsa vozlišča so bila neposredno dosegljiva med sabo, kar modelira povezavo vozlišč na lokalnem omrežju brez vmesnih filtrov. Pognali smo 10, 30, 50, 80, 100, 200, 300, 500, 800 in 1000 instanc testne aplikacije, ter preverili kaj se nahaja v usmerjevalnih tabelah in kakšne povezave so se vzpostavile med instancami.

Vozlišče	Vedro	LCP
ed61c389c36d8d04aa3f4b92ac6de6a93eddba5d	0	4
ea10d0fe36401fb4fadca7effd0b3bfb0214ed99	0	4
f57180c736cc2609371be0978bbfe4ef6f2c9b51	0	3
b6ae10ffb7c2378f1e8728fa7a66c84891269f0e	0	1
b7867576c8dbd7e483e4b36c85a5958e7846a9ce	0	1
6ae07815d45f42ac9323c8b5ee9d6b80eb323160	0	0
48f2c921158ebc2bdfe622e0770a20c847d415e7	0	0
568412052177ec0e024e733c8cf38437f3a354a9	0	0
09a73cb5edad646af61abf7de1ba5227f695e126	0	0

Tabela 5.1: Prikaz usmerjevalne tabele za vozlišče `e61e...7f2` pri 10ih vozliščih.

Pri vseh meritvah smo uporabili parameter redundance posameznega k-vedra $k = 10$, enakomerno razpršenost po prostoru identifikatorjev pa je zagotavljala certifikatna avtoriteta s pomočjo naključnega generiranja identifikatorjev pri izdajanju certifikatov. Vsa vozlišča so poganjala identično programsko kodo, kar je v duhu s principom komunikacije med enakovrednimi vozlišči (oz. vsak-z-vsakim, angl. peer-to-peer).

5.2 Rezultati

Pri testu za vsako izmed števil vozlišč smo shranili usmerjevalne tabele nekaterih naključno izbranih vozlišč, da bi tako prikazali kako se usmerjanje obnaša v praksi.

Začeli smo z desetimi vozlišči in pogledali vsebino usmerjevalne tabele enega izmed njih (tabela 5.1). Takoj opazimo, da se vsa vozlišča nahajajo v k-vedru z indeksom 0 in da je torej v uporabi samo eno k-vedro. To je v skladu z dejstvom, da se vedra razdeljujejo šele, ko so polna, mi pa za testiranje uporabljamo faktor redundance $k = 10$, kar pa ravno dosega naše število vozlišč. V tabeli smo prikazali tudi polje *LCP* (angl. longest common prefix), ki nam pove število bitov z leve, ki jih ima identifikator v tabeli skupnih z identifikatorjem lokalnega vozlišča. To polje je v resnici neposreden odraz ciljnega k-vedra, kot bomo videli v nadaljevanju.

Naslednje testiranje je potekalo pri 30 vozliščih, kjer smo kot rezultat dobili 2 k-vedri, v vsakem pa je bilo po 10 vozlišč. Če dobro pogledamo usmerjevalne tabele (tabela 5.2), lahko opazimo zelo podobno razporeditev v k-vedru z indeksom 0. Drugačno je vedro z indeksom 1, kjer imajo vsi vnosi z lokalnim identifikatorjem skupen (polje *LCP*) vsaj 1 bit. To lepo ilustrira način organi-

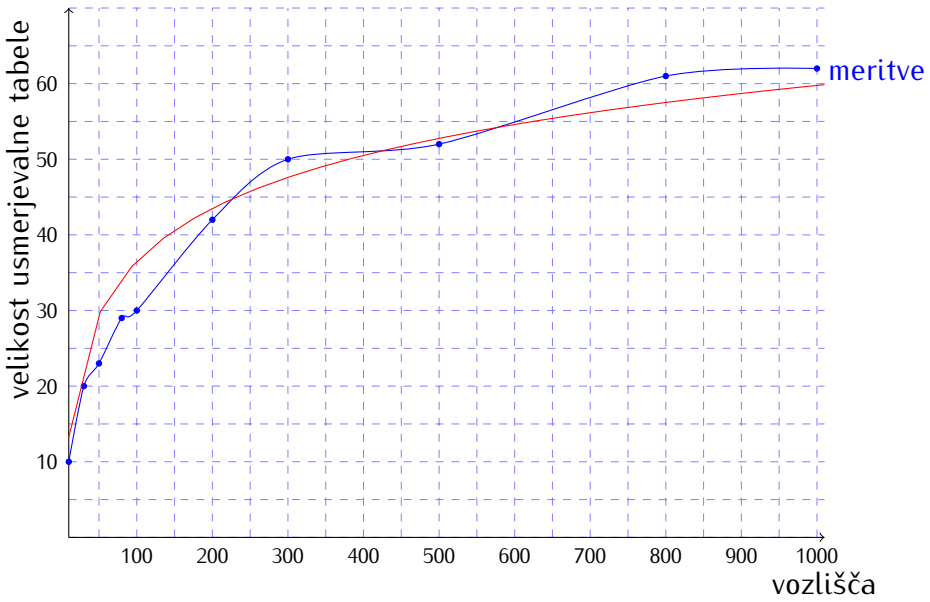
Vozlišče	Vedro	LCP
ed61c389c36d8d04aa3f4b92ac6de6a93eddba5d	0	4
ea10d0fe36401fb4fadca7effd0b3bfb0214ed99	0	4
f57180c736cc2609371be0978bbfe4ef6f2c9b51	0	3
b6ae10ffb7c2378f1e8728fa7a66c84891269f0e	0	1
b7867576c8dbd7e483e4b36c85a5958e7846a9ce	0	1
8ecc66b94160a4bcb80b11f546d31e7bb605d1ff	0	1
64a0b71526eb282bedee5b2e47fc53e61d3b174b	0	0
619fd323cfee6f41be5b9af1e15b10cf20b85597	0	0
6d4e89f7ca39620ba95f0b1b485f5727f915c81a	0	0
6ae07815d45f42ac9323c8b5ee9d6b80eb323160	0	0
e30f0484a4b2fe1d6b3e0cbda184aa1ac0c58942	1	5
f125414e5e1b40a066d31c7000a1950ea836e503	1	3
a637c5035b44d579770761669a3014d79fecd4e6	1	1
ae52f86746cc102212359d09505902782d2f88b9	1	1
b4dfd0987a1a85528ceb9f7246caa39d13df4def	1	1
b4c8d94fbb18009452a575b8c97a77dc2cb2a086	1	1
80ed7d73f1329fe317e59afbe9e3a34de2e184ad	1	1
96f5caaa8b2c7f6d5c98a03bc934a3c99856c2f4	1	1
92c5262c0aef183f65b3fe31e74fe2c294ebea06	1	1

Tabela 5.2: Prikaz usmerjevalne tabele za vozlišče `e61e...7f2` pri 30ih vozliščih.

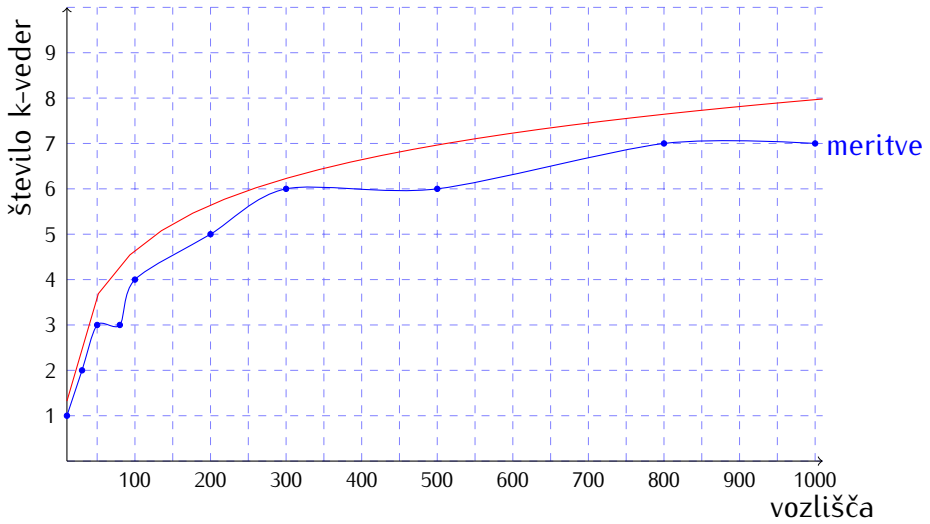
zacije k-veder, kjer imajo višja vedra vedno več skupnih bitov z identifikatorjem lokalnega vozlišča in so posledično tudi vedno bolj prazna (zaradi naključne razporeditve po prostoru je vedno manj kandidatov, ki bi lahko zapolnili visoko ležeča vedra, saj verjetnost za ujemanje strmo pada z večanjem indeksa). Predpostavka na kateri temelji razteglјivost našega prekrivnega omrežja je, da ker je za vedra z višjimi indeksi vedno težje najti kandidate, bo z večanjem števila vozlišč število veder raslo počasi. To predpostavko smo z meritvami v nadaljevanju tudi potrdili.

Če dobro pogledamo tabelo 5.2 lahko opazimo, da se v njej nahaja zgolj 20 vozlišč, čeprav jih je v celotnem prekrivnem omrežju 30. Usmerjevalni algoritem deluje kljub temu, da usmerjevalna tabela ne vsebuje popolne informacije, seveda na račun povečanja števila korakov pri odkrivanju ciljnih kontaktnih podatkov. To je ključna lastnost, ki nam zagotavlja podlinearno (v resnici logaritemsko) razteglјivost, lepo pa je vidna pri povečevanju števila vozlišč.

Pri večjem številu vozlišč bi bil izpis usmerjevalnih tabel nesmiselen, tako da smo opazovali zgolj rast števila vnosov v usmerjevalnih tabelah (slika 5.3)



Slika 5.3: Prikaz skalabilnosti usmerjevalne tabele (št. vseh vnosov) pri $k = 10$ in rasti števila vozlišč do 1000. Modro so označene meritve nad implementacijo našega ogrodja, rdeče je funkcija $7 * \log_2 n - 10$ za primerjavo hitrosti rasti.



Slika 5.4: Prikaz skalabilnosti usmerjevalne tabele (št. k-veder) pri $k = 10$ in rasti števila vozlišč do 1000. Modro so označene meritve nad implementacijo našega ogrodja, rdeče je funkcija $\log_2 n - 2$ za primerjavo hitrosti rasti.

ter rast števila vseh k-veder v tabelah (slika 5.4) posameznega vozlišča. Iz omenjenih meritev je lepo razvidna logaritemska razteglјivost, kar nam potrjuje eno izmed lepih lastnosti našega ogrodja.

5.3 Ugotovitve testiranja

Pri testiranju se je izkazalo, da ogrodje deluje pravilno in zanesljivo, manjše ugotovljene napake pa so bile tekom testiranja odpravljene. Odzivnost ogrodja je bila visoka, pokazali pa smo, da je tudi v praksi razteglјivost sistema enaka teoretični, torej smo zadostili zahtevi pričakovane razteglјivosti, ki smo si jo postavili pred načrtovanjem. Ogrodje posamezno vozlišče obremenjuje zanemarljivo, tako glede procesorskih kot tudi glede pomnilniških virov. S tem smo dokazali vse zahtevane lastnosti, ki jih lahko preizkusimo v testnem, simuliranem okolju. Zato lahko potrdimo, da je komunikacijsko ogrodje pripravljeno za preizkus v realni aplikaciji, ki jo uporablja veliko število razpršenih vozlišč. Menimo, da bo ogrodje UNISPHERE tudi v takem "resničnem svetu" pokazalo, da ustreza zahtevam, ki smo si jih zastavili.

Poglavje 6

Sklep

V diplomski nalogi smo predstavili tako teoretično zasnovo kot tudi lastno praktično implementacijo komunikacijskega ogrodja za porazdeljene sisteme, ki temelji na usmerjanju sporočil z uporabo prekrivnih omrežij, zgrajenih po principu vsak-z-vsakim.

6.1 Zaključne ugotovitve

Po teoretični analizi in testiranju implementacije našega ogrodja smo prišli do nekaterih ugotovitev, ki jih na tem mestu kritično predstavimo:

1. Ogrodje se je izkazalo kot enostavno za uporabo za razvijalca (z malo kode je mogoče hitro zgraditi delujoč prototip), po (omejenih) meritvah in primerjavo le-teh s preprostim modelom pa se je izkazalo tudi kot razteglјivo. To potrjuje našo predpostavko iz uvoda in odpira možnosti uporabe na področjih kot so omrežno upravljanje, upravljanje konfiguracije, porazdeljeni avtentikacijski sistemi, upravljanje storitev in drugih.

2. Vendar se je na tem mestu potrebno zavedati, da ogrodje ni *srebrna krogla*, ki lahko reši vse probleme v komunikacijah. Trenutno ima nekatere omejitve, ki otežujejo uporabo v sistemih z veliko organizacijami (predpostavka o centralizirani certifikatni avtoriteti). Vendar te omejitve hkrati predstavljajo priložnosti za dodatne raziskave in za razvoj dodatnih funkcionalnosti.

3. Meritve, ki smo jih izvedli še zdaleč niso izčrpne. Za boljšo primerljivost med implementacijami bi bilo dobro razviti močno in odprto orodje, ki bi omogočalo simulacijo poljubnih topologij in omrežnih pogojev tudi v

praksi. Prav tako bi bilo dobro imeti v samem ogrodju UNISPHERE tudi možnost živega spremljanja delovanja prekrivnega omrežja.

6.2 Nadaljnje delo

Zasnovana arhitektura in njena implementacija torej predstavljata dobro osnovo, ki pa ji do polne uporabne vrednosti manjka še nekaj detajlov. Nekateri izmed teh detajlov zahtevajo nadaljnje raziskave, saj določeni problemi, predvsem tisti, ki zadevajo varnost in porazdeljeno zaupanje v prekrivnih omrežjih, še niti v teoriji niso popolnoma rešeni. Nekaj takšnih nerešenih problemov z bolj teoretičnega področja je naslednjih:

- **Kompaktno usmerjanje:** V našem diplomskem delu smo predstavili usmerjanje s pomočjo strukturiranih usmerjevalnih protokolov, ki nam ne dajejo zagotovil o raztegu na nivoju IP. Zgolj na hitro pa smo se v teoretičnem pregledu dotaknili druge družine raztegljivih usmerjevalnih protokolov, kompaktnega usmerjanja [33], ki za razliko od prvih ne zahteva strukturiranih topologij in zagotavlja navzgor omejen razteg ob prostorski kompleksnosti $O(\sqrt{n \log n})$ na vozlišče. Prvi porazdeljeni protokoli (za razliko od centraliziranih algoritmov) za uspešno kompaktno usmerjanje na podlagi nehierarhičnih identifikatorjev so se začeli pojavljati šele v zadnjem času.

- **Porazdeljeno upravljanje zaupanja v sistemih P2P:** Kot smo omenili v poglavju o varnosti v prekrivnih omrežjih, je varovanje le-teh pri uporabi tradicionalnih protokolov zelo težavno. Ta težavnost izhaja iz predpostavk o homogenosti zaupanja vozlišč, ki v resničnem okolju niso upravičene. Na hitro smo omenili hierarhične izpeljanke strukturiranih prekrivnih omrežij, ki omogočajo določeno stopnjo avtonomije in predlagali njihovo uporabo v *avtonomnih sistemih zaupanja*. Potrebno bo povezati znanje s področja porazdeljenega zaupanja s tehnologijo usmerjanja v prekrivnih omrežjih.

Drugi možni dodatki so popolnoma inženirske narave, nekatere je potrebno preizkusiti in ugotoviti do kolikšne mere bi bili uporabni, druge pa je potrebno zgolj integrirati v ogrodje. Ker je ogrodje zastavljeno modularno, to ne bi smelo biti preveč težavno. Nekatere izmed rešitev, ki bi jih bilo smiselno preučiti, so:

- **CurveCP namesto TLS/TCP:** CurveCP [5] je novejši kriptografski protokol za zagotavljanje zaupnosti in integritete v internetnih komunikacijah. Po svoji vlogi je enakovreden protokolu TLS, ki uporablja certifikate X.509

in katerega uporabljamo tudi v našem ogrodju UNISPHERE. Zanimivo bi bilo preučiti možnost uporabe CurveCP, saj le-ta obeta hitrejšo vzpostavitev povezave, katero doseže s kompaktnim protokolom (v resnici zamenja tudi obstoječ TCP) in z uporabo hitrejše kriptografske funkcije, ki temelji na eliptični krivulji `Curve25519`.

- **Prečkanje prevajalnikov omrežnih naslovov (angl. NAT traversal):** Pri vzpostavljanju povezav skozi prevajalnike omrežnih naslovov se v opisani arhitekturi zanašamo zgolj na tuneliranje skozi druga vozlišča. Kot že omenjeno, bi bila najboljša kombinacija te tehnike s tehnikami prečkanja prevajalnikov omrežnih naslovov [13].

- **Imenske storitve:** V našem komunikacijskem ogrodju trenutno vse poteka neposredno preko identifikatorjev vozlišč. V določenih primerih bi bila bolj smiselna uporaba imenskih identifikatorjev, katere je nato potrebno preslikati v ustrezne identifikatorje. V ta namen bi lahko uporabili kar prekrivno omrežje in njegovo porazdeljeno zgoščevalno tabelo, paziti je treba zgolj na performančne in varnostne vidike.

- **Spremljanje delovanja omrežja:** Zelo pomembno pri razvoju porazdeljenih aplikacij (in tudi pri razvoju ogrodja samega) je živo spremljanje delovanja omrežja. Zelo dobrodošlo bi bilo torej razširiti ogrodje z generično rešitvijo za spremljanje različnih performančnih in drugih kazalcev, ki bi jih lahko nato razvijalec ali administrator opazoval iz kateregakoli vozlišča v omrežju.

Prihodnost komunikacij je porazdeljena izmenjava informacij med neodvisnimi entitetami, ki imajo med sabo vzpostavljene različne relacije zaupanja. Realizacija takšnih sistemov ni preprosto opravilo, vendar z gradnjo arhitektur, kot je naša, takšno prihodnost prinašamo bližje k realizaciji po en korak naenkrat.

Literatura

[1] Information technology - Portable Operating System Interface (POSIX) Operating System Interface (POSIX). *ISO/IEC/IEEE 9945 (First edition 2009-09-15)*, 15 2009.

[2] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, in M. Thorup. Compact name-independent routing with minimum stretch. *ACM Transactions on Algorithms (TALG)*, 4(3):1–12, 2008.

[3] R. Arends, R. Austein, M. Larson, D. Massey, in S. Rose. Protocol Modifications for the DNS Security Extensions, mar 2005. Updated by RFCs 4470, 6014. Dostopno na `http://www.ietf.org/rfc/rfc4035.txt`.

[4] M. S. Artigas, P. G. Lopez, J. P. Ahullo, in A. F. G. Skarmeta. Cyclone: A novel design schema for hierarchical dhts. *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, strani 49–56, 2005.

[5] Daniel J. Bernstein. CurveCP: Usable security for the Internet, Dec 2010. Dostopno na `http://curvecp.org/`.

[6] M. Castro, P. Druschel, A. M. Kermarrec, in A. I. T. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489–1499, 2002.

[7] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, in Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, December 2002. Dostopno na `http://doi.acm.org/10.1145/844128.844156`.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, in S. Shenker. Making gnutella-like p2p systems scalable. Objavljeno v *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, strani 407–418. ACM, 2003.

[9] M. Cooper, Y. Dzambasow, P. Hesse, S. Joseph, in R. Nicholas. Internet X.509 Public Key Infrastructure: Certification Path Building, sep 2005. Dostopno na `http://www.ietf.org/rfc/rfc4158.txt`.

[10] T. Dierks in E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2, aug 2008. Updated by RFCs 5746, 5878. Dostopno na `http://www.ietf.org/rfc/rfc5246.txt`.

[11] J. Feigenbaum, M. Blaze, in J. Lacy. Decentralized trust management. Objavljeno v *Proceedings of the IEEE Conference on Security and Privacy*, strani 164–173. IEEE Computer Society, 1996.

[12] B. Fenner, M. Handley, H. Holbrook, in I. Kouvelas. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised), aug 2006. Updated by RFCs 5059, 5796. Dostopno na `http://www.ietf.org/rfc/rfc4601.txt`.

[13] B. Ford, P. Srisuresh, in D. Kegel. Peer-to-peer communication across network address translators. Objavljeno v *USENIX Annual Technical Conference*, 2005.

[14] Bryan Ford. *UIA: A Global Connectivity Architecture for Mobile Personal Devices*. Doktorska disertacija, Massachusetts Institute of Technology, 2008.

[15] Michael J. Freedman, Karthik Lakshminarayanan, Sean Rhea, in Ion Stoica. Non-transitive connectivity and DHTs. Objavljeno v *Proceedings of the 2nd conference on Real, Large Distributed Systems - Volume 2*, WORLDS'05, strani 55–60, Berkeley, CA, USA, 2005. WORLDS '05. Dostopno na `http://portal.acm.org/citation.cfm?id=1251522.1251532`.

[16] P. Ganesan, K. Gummadi, in H. Garcia-Molina. Canon in G major: designing DHTs with hierarchical structure. *Proceedings, 24th International Conference on Distributed Computing Systems*, strani 263–272, 2004.

[17] P. Ganesan, Q. Sun, in H. Garcia-Molina. Yappers: A peer-to-peer lookup service over arbitrary topology. Objavljeno v *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, del 2, strani 1250–1260. IEEE, 2003.

[18] Phillip Hallam-Baker. The Recent RA Compromise, mar 2011. Dostopno na `http://blogs.comodo.com/it-security/data-security/the-recent-ra-compro%mise/`.

[19] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, in A. Wolman. Skipnet: A scalable overlay network with practical locality properties. Objavljeno v *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems-Volume 4*, stran 9. USENIX Association, 2003.

[20] S. Jain, Y. Chen, in Z. L. Zhang. VIRO: A Plug and Play, Scalable, Robust and Namespace Independent Virtual Id Routing for Future Networks. Technical report, 2009. Dostopno na `http://networking.cs.umn.edu/veil/viro.pdf`.

[21] S. Josefsson. The Base16, Base32, and Base64 Data Encodings, oct 2006. Dostopno na `http://www.ietf.org/rfc/rfc4648.txt`.

[22] S. Kent in K. Seo. Security Architecture for the Internet Protocol, dec 2005. Updated by RFC 6040. Dostopno na `http://www.ietf.org/rfc/rfc4301.txt`.

[23] C. Lesniewski-Laas in M. F. Kaashoek. Whanau: A sybil-proof distributed hash table. Objavljeno v *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, stran 8. USENIX Association, 2010.

[24] Petar Maymounkov in David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. Objavljeno v *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, strani 53–65, London, UK, 2002. Springer-Verlag. Dostopno na `http://portal.acm.org/citation.cfm?id=646334.687801`.

[25] A. Mislove in P. Druschel. Providing administrative control and autonomy in structured peer-to-peer overlays. *Peer-to-Peer Systems III*, strani 162–172, 2005.

[26] J. Moy. OSPF Version 2, apr 1998. Updated by RFC 5709. Dostopno na `http://www.ietf.org/rfc/rfc2328.txt`.

[27] C. G. Plaxton, R. Rajaraman, in A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32(3):241–280, 1999.

[28] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[29] S. Ratnasamy, P. Francis, M. Handley, R. Karp, in S. Shenker. A scalable content-addressable network. Objavljeno v *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, strani 161–172. ACM, 2001.

[30] A. Rowstron in P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Objavljeno v *Middleware 2001*, strani 329–350. Springer, 2001.

[31] Atul Singh, Tsuen-wan Ngan Ngan, Peter Druschel, in Dan S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. Objavljeno v *In IEEE INFOCOM*, 2006.

[32] Atul Singh, Tsuen-wan Ngan Ngan, Peter Druschel, in Dan S. Wallach. Implementation and Evaluation of Secure Routing Primitives. Technical report, Rice University, Jan 2006.

[33] A. Singla, P. Godfrey, K. Fall, G. Iannaccone, in S. Ratnasamy. Scalable routing on flat names. Objavljeno v *Proceedings of the 6th International Conference*, stran 20. ACM, 2010.

[34] E. Sit in R. Morris. Security considerations for peer-to-peer distributed hash tables. *Peer-to-Peer Systems*, strani 261–269, 2002.

[35] W. Stallings. PGP web of trust. *Byte*, 20(2):161–162, 1995.

[36] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, in H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. Objavljeno v *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, strani 149–160. ACM, 2001.

[37] G. Urdaneta, G. Pierre, in M. V. Steen. A survey of DHT security techniques. *ACM Computing Surveys (CSUR)*, 43(2):8, 2011.

[38] Dan S. Wallach. A survey of peer-to-peer security issues. Objavljeno v *Proceedings of the 2002 Mext-NSF-JSPS international conference on Software security: theories and systems*, ISSS'02, strani 42–57, Berlin, Heidelberg, 2003. Springer-Verlag. Dostopno na `http://portal.acm.org/citation.cfm?id=1765533.1765539`.

[39] Anastasios Zafeiropoulos, Panagiotis Gouvas, Athanassios Liakopoulos, Gregoris Mentzas, in Nikolas Mitrou. NEURON: Enabling Autonomicity in Wireless Sensor Networks. *Sensors*, 10(5):5233–5262, 2010. Dostopno na `http://www.mdpi.com/1424-8220/10/5/5233/`.

[40] B. Y. Zhao, J. D. Kubiatowicz, in A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Computer*, 74:46, 2001.